

# 48th International Symposium on Mathematical Foundations of Computer Science

MFCS 2023, August 28 to September 1, 2023, Bordeaux, France

Edited by

Jérôme Leroux

Sylvain Lombardy

David Peleg



*Editors*

**Jérôme Leroux** 

LaBRI, Université Bordeaux, CNRS, Bordeaux-INP, Talence, France  
jerome.leroux@labri.fr

**Sylvain Lombardy** 

LaBRI, Université Bordeaux, CNRS, Bordeaux-INP, Talence, France  
sylvain.lombardy@labri.fr

**David Peleg** 

Weizmann Institute of Science, Rehovot, Israel  
david.peleg@weizmann.ac.il

*ACM Classification 2012*

Theory of computation; Mathematics of computing

**ISBN 978-3-95977-292-1**

*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/978-3-95977-292-1>.

*Publication date*

August, 2023

*Bibliographic information published by the Deutsche Nationalbibliothek*

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

*License*

This work is licensed under a Creative Commons Attribution 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.MFCS.2023.0

ISBN 978-3-95977-292-1

ISSN 1868-8969

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

## LIPICs – Leibniz International Proceedings in Informatics

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

### *Editorial Board*

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

**ISSN 1868-8969**

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



## ■ Contents

Preface	
<i>Jérôme Leroux, Sylvain Lombardy, and David Peleg</i> .....	0:xi
Organization	
.....	0:xiii

### Invited Talks

Exploring the Space of Colourings with Kempe Changes	
<i>Marthe Bonamy</i> .....	1:1–1:2
Online Algorithms with Predictions	
<i>Joan Boyar</i> .....	2:1–2:2
Modern Parallel Algorithms	
<i>Artur Czumaj</i> .....	3:1–3:2
Algebraic Reasoning for (Un)Solvable Loops	
<i>Laura Kovács</i> .....	4:1–4:2
Sliding into the Future: Investigating Sliding Windows in Temporal Graphs	
<i>Nina Klobas, George B. Mertzios, and Paul G. Spirakis</i> .....	5:1–5:12

### Regular Papers

Roman Census: Enumerating and Counting Roman Dominating Functions on Graph Classes	
<i>Faisal N. Abu-Khzam, Henning Fernau, and Kevin Mann</i> .....	6:1–6:15
Counting Computations with Formulae: Logical Characterisations of Counting Complexity Classes	
<i>Antonis Achilleos and Aggeliki Chalki</i> .....	7:1–7:15
Recognizing $H$ -Graphs – Beyond Circular-Arc Graphs	
<i>Deniz Aġaoġlu Çaġırıcı, Onur Çaġırıcı, Jan Derbisz, Tim A. Hartmann, Petr Hliněný, Jan Kratochvíl, Tomasz Krawczyk, and Peter Zeman</i> .....	8:1–8:14
Descriptive Complexity for Distributed Computing with Circuits	
<i>Veeti Ahvonen, Damian Heiman, Lauri Hella, and Antti Kuusisto</i> .....	9:1–9:15
Solving Irreducible Stochastic Mean-Payoff Games and Entropy Games by Relative Krasnoselskii–Mann Iteration	
<i>Marianne Akian, Stéphane Gaubert, Ulysse Naepels, and Basile Terver</i> .....	10:1–10:15
The Geometry of Reachability in Continuous Vector Addition Systems with States	
<i>Shaull Almagor, Arka Ghosh, Tim Leys, and Guillermo A. Pérez</i> .....	11:1–11:13
Competitive Search in the Line and the Star with Predictions	
<i>Spyros Angelopoulos</i> .....	12:1–12:15

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg

Leibniz International Proceedings in Informatics



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

Rényi-Ulam Games and Online Computation with Imperfect Advice <i>Spyros Angelopoulos and Shahin Kamali</i> .....	13:1–13:15
Multivariate to Bivariate Reduction for Noncommutative Polynomial Factorization <i>Vikraman Arvind and Pushkar S. Joglekar</i> .....	14:1–14:15
Entropic Risk for Turn-Based Stochastic Games <i>Christel Baier, Krishnendu Chatterjee, Tobias Meggendorfer, and Jakob Piribauer</i> .....	15:1–15:16
Speed Me up If You Can: Conditional Lower Bounds on Opacity Verification <i>Jiří Balun, Tomáš Masopust, and Petr Osíčka</i> .....	16:1–16:15
Separating Automatic Relations <i>Pablo Barceló, Diego Figueira, and Rémi Morvan</i> .....	17:1–17:15
On the Parameterized Complexity of Computing <i>st</i> -Orientations with Few Transitive Edges <i>Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, Giacomo Ortali, and Tommaso Piselli</i> .....	18:1–18:15
Distributed CONGEST Algorithm for Finding Hamiltonian Paths in Dirac Graphs and Generalizations <i>Noy Biton, Reut Levi, and Moti Medina</i> .....	19:1–19:14
Locality Theorems in Semiring Semantics <i>Clotilde Bizière, Erich Grädel, and Matthias Naaf</i> .....	20:1–20:15
A Characterisation of Functions Computable in Polynomial Time and Space over the Reals with Discrete Ordinary Differential Equations: Simulation of Turing Machines with Analytic Discrete ODEs <i>Manon Blanc and Olivier Bournez</i> .....	21:1–21:15
MaxCut Above Guarantee <i>Ivan Bliznets and Vladislav Epifanov</i> .....	22:1–22:14
Cryptanalysis of a Generalized Subset-Sum Pseudorandom Generator <i>Charles Bouillaguet, Florette Martinez, and Damien Vergnaud</i> .....	23:1–23:15
The Compositional Structure of Bayesian Inference <i>Dylan Braithwaite, Jules Hedges, and Toby St Clere Smithe</i> .....	24:1–24:15
Deterministic Constrained Multilinear Detection <i>Cornelius Brand, Viktoriia Korchemna, and Michael Skotnica</i> .....	25:1–25:14
Rational Verification for Nash and Subgame-Perfect Equilibria in Graph Games <i>Léonard Brice, Jean-François Raskin, and Marie van den Bogaard</i> .....	26:1–26:15
On Property Testing of the Binary Rank <i>Nader H. Bshouty</i> .....	27:1–27:14
Short Definitions in Constraint Languages <i>Jakub Bulín and Michael Kompatscher</i> .....	28:1–28:15

The Online Simple Knapsack Problem with Reservation and Removability <i>Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, and Peter Rossmanith</i> .....	29:1–29:12
Parikh One-Counter Automata <i>Michaël Cadilhac, Arka Ghosh, Guillermo A. Pérez, and Ritam Raha</i> .....	30:1–30:15
Modification Problems Toward Proper (Helly) Circular-Arc Graphs <i>Yixin Cao, Hanchun Yuan, and Jianxin Wang</i> .....	31:1–31:14
Isometric Path Complexity of Graphs <i>Dibyayan Chakraborty, Jérémie Chalopin, Florent Foucaud, and Yann Vaxès</i> .....	32:1–32:14
Support Size Estimation: The Power of Conditioning <i>Diptarka Chakraborty, Gunjan Kumar, and Kuldeep S. Meel</i> .....	33:1–33:13
Query Complexity of Search Problems <i>Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan</i> .....	34:1–34:15
Tight Algorithmic Applications of Clique-Width Generalizations <i>Vera Chekan and Stefan Kratsch</i> .....	35:1–35:15
An Iterative Approach for Counting Reduced Ordered Binary Decision Diagrams <i>Julien Clément and Antoine Genitrini</i> .....	36:1–36:15
Inductive Continuity via Brouwer Trees <i>Liron Cohen, Bruno da Rocha Paiva, Vincent Rahli, and Ayberk Tosun</i> .....	37:1–37:16
Approximating the Value of Energy-Parity Objectives in Simple Stochastic Games <i>Mohan Dantam and Richard Mayr</i> .....	38:1–38:15
Dynamic Planar Embedding Is in DynFO <i>Samir Datta, Asif Khan, and Anish Mukherjee</i> .....	39:1–39:15
Universality and Forall-Exactness of Cost Register Automata with Few Registers <i>Laure Daviaud and Andrew Ryzhikov</i> .....	40:1–40:15
Relaxed Core Stability for Hedonic Games with Size-Dependent Utilities <i>Tom Demeulemeester and Jannik Peters</i> .....	41:1–41:14
Recontamination Helps a Lot to Hunt a Rabbit <i>Thomas Dissaux, Foivos Fioravantes, Harmender Gahlawat, and Nicolas Nisse</i> ...	42:1–42:14
String Diagrammatic Trace Theory <i>Matthew Earnshaw and Paweł Sobociński</i> .....	43:1–43:15
Upward Translation of Optimal and P-Optimal Proof Systems in the Boolean Hierarchy over NP <i>Fabian Egidy, Christian Glaßer, and Martin Herold</i> .....	44:1–44:15
Finding a Highly Connected Steiner Subgraph and its Applications <i>Eduard Eiben, Diptapriyo Majumdar, and M. S. Ramanujan</i> .....	45:1–45:15
FPT Approximation and Subexponential Algorithms for Covering Few or Many Edges <i>Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, and Tomohiro Koana</i> .....	46:1–46:8

Graph Connectivity with Noisy Queries <i>Dimitris Fotakis, Evangelia Gergatsouli, Charilaos Pipis, Miltiadis Stouras, and Christos Tzamos</i> .....	47:1–47:14
Positive Data Languages <i>Florian Frank, Stefan Milius, and Henning Urbat</i> .....	48:1–48:15
Parameterized Analysis of the Cops and Robber Game <i>Harmender Gahlawat and Meirav Zehavi</i> .....	49:1–49:17
An FPT Algorithm for Spanning Trees with Few Branch Vertices Parameterized by Modular-Width <i>Luisa Gargano and Adele A. Rescigno</i> .....	50:1–50:15
Depth-3 Circuits for Inner Product <i>Mika Göös, Ziyi Guan, and Tiberiu Mosnoi</i> .....	51:1–51:12
On Polynomial-Time Decidability of $k$ -Negations Fragments of FO Theories (Extended Abstract) <i>Christoph Haase, Alessio Mansutti, and Amaury Pouly</i> .....	52:1–52:14
The Covering Canadian Traveller Problem Revisited <i>Niklas Hahn and Michalis Xeferis</i> .....	53:1–53:12
On the Complexity of Computing Time Series Medians Under the Move-Split-Merge Metric <i>Jana Holznigenkemper, Christian Komusiewicz, Nils Morawietz, and Bernhard Seeger</i> .....	54:1–54:15
Fixed-Parameter Algorithms for Fair Hitting Set Problems <i>Tanmay Inamdar, Lawqueen Kanesh, Madhumita Kundu, Nidhi Purohit, and Saket Saurabh</i> .....	55:1–55:14
Parameterized Approximation Scheme for Feedback Vertex Set <i>Satyabrata Jana, Daniel Lokshantov, Soumen Mandal, Ashutosh Rai, and Saket Saurabh</i> .....	56:1–56:15
Complexity Framework for Forbidden Subgraphs III: When Problems Are Tractable on Subcubic Graphs <i>Matthew Johnson, Barnaby Martin, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen</i> .....	57:1–57:15
Polynomial-Delay Enumeration of Large Maximal Common Independent Sets in Two Matroids <i>Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiko Wasa</i> .....	58:1–58:14
Formalizing Hyperspaces for Extracting Efficient Exact Real Computation <i>Michal Konečný, Sewon Park, and Holger Thies</i> .....	59:1–59:16
Set Semantics for Asynchronous TeamLTL: Expressivity and Complexity <i>Juha Kontinen, Max Sandström, and Jonni Virtema</i> .....	60:1–60:14
Parameterized Complexity of Domination Problems Using Restricted Modular Partitions <i>Manuel Lafond and Weidong Luo</i> .....	61:1–61:14

Parameterized Max Min Feedback Vertex Set <i>Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis</i> .....	62:1–62:15
Distributed Merlin-Arthur Synthesis of Quantum States and Its Applications <i>François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura</i> .....	63:1–63:15
Dichotomies for Maximum Matching Cut: $H$ -Freeness, Bounded Diameter, Bounded Radius <i>Felicia Lucke, Daniël Paulusma, and Bernard Ries</i> .....	64:1–64:15
A Weyl Criterion for Finite-State Dimension and Applications <i>Jack H. Lutz, Satyadev Nandakumar, and Subin Pulari</i> .....	65:1–65:16
On the Complexity Dichotomy for the Satisfiability of Systems of Term Equations over Finite Algebras <i>Peter Mayr</i> .....	66:1–66:12
Parallel Enumeration of Parse Trees <i>Margarita Mikhelson and Alexander Okhotin</i> .....	67:1–67:14
Spartan Bipartite Graphs Are Essentially Elementary <i>Neeldhara Misra and Saraswati Girish Nanoti</i> .....	68:1–68:15
On the Finite Variable-Occurrence Fragment of the Calculus of Relations with Bounded Dot-Dagger Alternation <i>Yoshiki Nakamura</i> .....	69:1–69:15
Effective Continued Fraction Dimension Versus Effective Hausdorff Dimension of Reals <i>Satyadev Nandakumar, Akhil S, and Prateek Vishnoi</i> .....	70:1–70:15
On the Expressive Power of Regular Expressions with Backreferences <i>Taisei Nogami and Tachio Terauchi</i> .....	71:1–71:15
OBDD(Join) Proofs Cannot Be Balanced <i>Sergei Ovcharov</i> .....	72:1–72:13
Lower Bounds for Choiceless Polynomial Time via Symmetric XOR-Circuits <i>Benedikt Pago</i> .....	73:1–73:15
A Super-Polynomial Separation Between Resolution and Cut-Free Sequent Calculus <i>Theodoros Papamakarios</i> .....	74:1–74:15
Realizing Finitely Presented Groups as Projective Fundamental Groups of SFTs <i>Léo Paviet Salomon and Pascal Vanier</i> .....	75:1–75:15
Deciding Predicate Logical Theories Of Real-Valued Functions <i>Stefan Ratschan</i> .....	76:1–76:15
A Polynomial-Time Algorithm for MCS Partial Search Order on Chordal Graphs <i>Guozhen Rong, Yongjie Yang, and Wenjun Li</i> .....	77:1–77:15
Probabilistic Input-Driven Pushdown Automata <i>Alex Rose and Alexander Okhotin</i> .....	78:1–78:14

Counting Homomorphisms from Hypergraphs of Bounded Generalised Hypertree Width: A Logical Characterisation <i>Benjamin Scheidt and Nicole Schweikardt</i> .....	79:1–79:15
Dynamic Constant Time Parallel Graph Algorithms with Sub-Linear Work <i>Jonas Schmidt and Thomas Schwentick</i> .....	80:1–80:15
On the Work of Dynamic Constant-Time Parallel Algorithms for Regular Tree Languages and Context-Free Languages <i>Jonas Schmidt, Thomas Schwentick, and Jennifer Todtshoef</i> .....	81:1–81:15
Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors <i>Tim Seppelt</i> .....	82:1–82:15
Decomposing Finite Languages <i>Daniel Alexander Spenner</i> .....	83:1–83:14
Dependent $k$ -Set Packing on Polymoids <i>Meng-Tsung Tsai, Shi-Chun Tsai, and Tsung-Ta Wu</i> .....	84:1–84:15
Exponential Lower Bounds for Threshold Circuits of Sub-Linear Depth and Energy <i>Kei Uchizawa and Haruki Abe</i> .....	85:1–85:15
Exact and Approximation Algorithms for Routing a Convoy Through a Graph <i>Martijn van Ee, Tim Oosterwijk, René Sitters, and Andreas Wiese</i> .....	86:1–86:15
Ordinal Measures of the Set of Finite Multisets <i>Isa Vialard</i> .....	87:1–87:15
Checking Presence Reachability Properties on Parameterized Shared-Memory Systems <i>Nicolas Waldburger</i> .....	88:1–88:15

## ■ Preface

The International Symposium on Mathematical Foundations of Computer Science (MFCS conference series) is a venue for high-quality original research in all branches of Theoretical Computer Science. MFCS is among the conferences with the longest history in the field – the first conference in the series was held already in 1972. For many years, the conference rotated between the Czech Republic, Slovakia, and Poland, but since 2013 it has expanded its realm and traveled around different countries in Europe. In 2023, at its 48th edition, MFCS was held in Bordeaux, France, on August 28th – September 1st.

This volume contains the invited contributions and the 84 contributed talks. The invited speakers were Marthe Bonamy (University of Bordeaux, France), Joan Boyar (University of Southern Denmark), Artur Czumaj (University of Warwick, UK), Laura Kovacs (TU Wien, Austria), and Paul Spirakis (university of Liverpool, UK).

The program committee of MFCS 2023 selected 84 papers out of the 209 submissions, with the authors of the submitted papers representing 38 countries. We express our deep gratitude to all the members of the program committee and the reviewers for their extensive reports and thorough discussions on the submissions' merits. We also warmly thank the invited speakers, as well as the authors of the submitted papers. MFCS proceedings have been published in the Dagstuhl/LIPIcs series since 2016. We thank Michael Wagner and the LIPIcs team for their kind help and support.

Jérôme Leroux  
Sylvain Lombardy  
David Peleg





# ■ Organization

## Program Committee

- Parosh Aziz Abdulla (Uppsala University)
- Akanksha Agrawal (Indian Institute of Technology Madras)
- Susanne Albers (Technical University of Munich)
- Josh Alman (Columbia University)
- Christel Baier (Technische Universität Dresden)
- Laurent Bienvenu (University of Bordeaux)
- Hans Bodlaender (Utrecht University)
- Olivier Bournez (Ecole Polytechnique)
- Véronique Bruyère (University of Mons)
- Ioannis Caragiannis (Aarhus University)
- Keerti Choudhary (Indian Institute of Technology Delhi)
- Thierry Coquand (University of Gothenburg)
- Stéphanie Delaune (Univiversité Rennes, CNRS, IRISA)
- Javier Esparza (Technische Universität München)
- Piotr Faliszewski (AGH University of Science and Technology)
- Michal Feldman (Tel-Aviv University)
- Paola Flocchini (University of Ottawa)
- Luisa Gargano (Università di Salerno)
- Leszek Gąsieniec (University of Liverpool)
- Jean Goubault-Larrecq (ENS Paris-Saclay)
- Petr Jancar (Palacky University)
- Lefteris M. Krousis (National and Kapodistrian University of Athens)
- Sándor Kisfaludi-Bak (Aalto University)
- Lukasz Kowalik (University of Warsaw)
- Daniel Král (Masaryk University)
- Rastislav Kráľovič (Comenius University)
- Ranko Lazic (University of Warwick)
- Jérôme Leroux, **co-Chair** (LaBRI, CNRS, Université de Bordeaux)
- Christof Löding (RWTH Aachen University)
- Pinyan Lu (Shanghai University of Finance and Economics)
- Elvira Mayordomo (Universidad de Zaragoza)
- Paolo Milazzo (University of Pisa)
- David Peleg, **co-Chair** (Weizmann Institute of Science)
- Simon Perdrix (Inria, Loria)
- Sophie Pinchinat (Université de Rennes)
- Jörg Rothe (Heinrich-Heine-Universität Düsseldorf)

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).  
Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg



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

- David Saulpic (IST Austria)
- Jiri Sgall (Charles University)
- Mahsa Shirmohammadi (CNRS, IRIF, Université Paris Cité)
- Michał Skrzypczak (University of Warsaw)
- Shay Solomon (Tel Aviv University)
- Dimitrios Thilikos (CNRS, LRIMM, Université de Montpellier)
- Patrick Totzke (University of Liverpool)
- Ashutosh Trivedi (University of Colorado Boulder)
- James Worrell (University of Oxford)
- Xiaoming Sun (Chinese Academy of Sciences)
- Meirav Zehavi (Ben-Gurion University)
- Marius Zimand (Towson University)
- Martin Zimmermann (Aalborg University)

### External Reviewers

- |                           |                         |                         |
|---------------------------|-------------------------|-------------------------|
| ■ Marek Adamczyk          | ■ Nicolas Bitar         | ■ Juhi Chaudhary        |
| ■ Avantika Agarwal        | ■ Ivan Bliznets         | ■ Yijia Chen            |
| ■ Hugo Akitaya            | ■ Achim Blumensath      | ■ Zongchen Chen         |
| ■ Eric Allender           | ■ Manuel Bodirsky       | ■ Pingan Cheng          |
| ■ Ioannis Anagnostides    | ■ Linus Boes            | ■ Dmitry Chistikov      |
| ■ Markus Anders           | ■ León Bohn             | ■ Ferdinando Cicalese   |
| ■ Spyros Angelopoulos     | ■ Benedikt Bollig       | ■ Lorenzo Clemente      |
| ■ Emmanuel Arrighi        | ■ Sougata Bose          | ■ Andrei Codreanu       |
| ■ Nathalie Aubrun         | ■ Cornelius Brand       | ■ Christian Coester     |
| ■ Miriam Backens          | ■ Roberto Bruni         | ■ Gennaro Cordasco      |
| ■ Jakub Balabán           | ■ Martin Bullinger      | ■ Timothée Corsini      |
| ■ Nikhil Balaji           | ■ Sam Buss              | ■ Mónika Csikós         |
| ■ A. R. Balasubramanian   | ■ Michaël Cadilhac      | ■ Radu Curticapean      |
| ■ Susobhan Bandopadhyay   | ■ Clément Canonne       | ■ Philipp Czerner       |
| ■ Corentin Barloy         | ■ Yixin Cao             | ■ Valentin Dardilhac    |
| ■ Nicolas Basset          | ■ Florent Capelli       | ■ Sandip Das            |
| ■ Balthazar Bauer         | ■ Titouan Carette       | ■ Joshua Daymude        |
| ■ Veronica Becher         | ■ Olivier Carton        | ■ Annalisa De Bonis     |
| ■ Dylan Bellier           | ■ Katrin Casel          | ■ Aldric Degorre        |
| ■ Rémy Belmonte           | ■ Katarina Cechlarova   | ■ Daniele Dell’Erba     |
| ■ Ulrich Berger           | ■ Pavol Cerny           | ■ Patrick Derbez        |
| ■ Martin Berglund         | ■ Sayak Chakrabarti     | ■ Jan Derbisz           |
| ■ Benjamin Bergougnoux    | ■ Dibyayan Chakraborty  | ■ Dariusz Dereniowski   |
| ■ Dietmar Berwanger       | ■ Aristotelis Chaniotis | ■ Krishnamoorthy Dinesh |
| ■ Olaf Beyersdorff        | ■ Prerona Chatterjee    | ■ Stefan Dobrev         |
| ■ Arpitha Prasad Bharathi | ■ Soumyottam Chatterjee | ■ Frank Drewes          |
|                           |                         | ■ Jérémy Dubut          |
|                           |                         | ■ Guillaume Ducoffe     |

- Arnaud Durand
- Franziska Eberle
- Eduard Eiben
- Marek Elias
- Angelo Fanelli
- John Fearnley
- Karl Fehrs
- Andreas Emil Feldmann
- Jiri Fiala
- Johannes K. Fichte
- Diego Figueira
- Nathanaël Fijalkow
- Emmanuel Filiot
- Aris Filos-Ratsikas
- Lukáš Folwarczny
- Fedor Fomin
- Chase Ford
- Marie Fortin
- Tim French
- Zhiguo Fu
- Harmender Gahlawat
- Ajinkya Gaikwad
- Jakub Gajarský
- Esther Galby
- Nicola Galesi
- Moses Ganardi
- Robert Ganian
- Francesco Gavazzo
- Arka Ghosh
- Archontia Giannopoulou
- Ioannis Giotis
- Rodolphe Giroudeau
- Michał Godziszewski
- Piotr Golovach
- Alexander Golovnev
- Roberta Gori
- Alexandre Goy
- Daniel Graça
- Julien Grange
- Mario Grobler
- Martin Gronemann
- Svyatoslav Gryaznov
- Pierre Guillon
- Frank Gurski
- Julian Gutierrez
- Roland Guttenberg
- Peter Habermehl
- Michel Habib
- Niklas Hahn
- Emmanuel Hainry
- Thekla Hamm
- Yassine Hamoudi
- Miika Hannula
- Kristoffer Arnsfelt Hansen
- Matthew Harrison-Trainor
- Hovhannes Harutyunyan
- Meike Hatzel
- Markus Hecher
- Klaus Heeger
- Irene Heinrich
- Nicolas Hermann
- Luisa Herrmann
- John M. Hitchcock
- Ullrich Hustadt
- Cameron Ibrahim
- Rasmus Ibsen-Jensen
- Christian Ikenmeyer
- Sungjin Im
- Tanmay Inamdar
- Gabriel Istrate
- Reijo Jaakkola
- Guilhem Jaber
- Bart Jacobs
- Lars Jaffke
- Tomáš Jakl
- Satyabrata Jana
- Łukasz Janeczko
- Emmanuel Jeandel
- Ismaël Jecker
- Zhile Jiang
- Bengt Jonsson
- Vincent Jugé
- Tomasz Jurdzinski
- Andrzej Kaczmarczyk
- Joanna Kaczmarek
- Gautam Kamath
- Alexis Kaporis
- Alexandr Kazda
- Yuping Ke
- George Kenison
- Donggyu Kim
- Kohei Kishida
- Bartek Klin
- Nina Klobas
- Dušan Knop
- Ivana Kolingerova
- Petr Kolman
- Christian Komusiewicz
- Spyros Kontogiannis
- Tuukka Korhonen
- Peter Kostolányi
- Alexander Kozachinskiy
- Jakub Kozik
- Laszlo Kozma
- S. Krishna
- Ariel Kulik
- Denis Kuperberg
- Orna Kupferman
- David Kutner
- Hoang La
- Michael Lampis
- Frédéric Lang
- Richard Lang
- Matthias Lanzinger
- Christian Laußmann
- Marijana Lazic
- Hoang-Oanh Le
- Jérémy Ledent
- Chuan-Min Lee
- Euiwoong Lee
- Louis Lemonnier
- Michael Levet
- Francesca Levi
- Alexander Lew
- Minming Li
- Moritz Lichter
- Bingkai Lin

- Jiabao Lin
- Yu-Yang Lin
- Andrzej Lingas
- John Livieratos
- Andrea Lucchini
- Michael Luttenberger
- Jack H. Lutz
- Aaron Lye
- Edita Macajova
- Jayakrishnan Madathil
- Nicolas Magaud
- Diptapriyo Majumdar
- Florin Manea
- David Manlove
- Alessio Mansutti
- Amaldev Manuel
- Mathieu Mari
- Andrea Marino
- Tomáš Masařík
- Tomas Masopust
- Simon Mauras
- Filippos Mavropoulos
- Ján Mazák
- Tobias Meggendorfer
- Nicole Megow
- Nikolaos Melissinos
- Ian Mertz
- Tony Metger
- Evi Micha
- Christian Michaux
- Marcus Michelen
- Vincent Michielini
- Pranabendu Misra
- Valia Mitsou
- Eiji Miyano
- Matthias Mnich
- Hendrik Molter
- Mikaël Monet
- Yoàv Montacute
- Benjamin Moore
- Amer Mouawad
- Martín Muñoz
- Vishnu Murali
- K. Narayan Kumar
- Varun Narayanan
- Daniel Neuen
- Eike Neumann
- Jim Newton
- Prajakta Nimbhorkar
- Klara Nosan
- Thomas Nowak
- Paul Nüsken
- André Nusser
- Pierre Ohlmann
- Naoto Ohsaka
- Karolina Okrasa
- Paulo Oliva
- Petr Osíčka
- Youssouf Oualhadj
- Charles Paperman
- Irene Parada
- Dana Pardubska
- Michal Parnas
- Paweł Parys
- Loïc Paulevé
- A Pavan
- Romain Péchoux
- Pierre-Marie Pédrot
- Kristyna Pekarkova
- Irena Penev
- Binghui Peng
- Anthony Perez
- Mateo Perez
- Geevarghese Philip
- Grzegorz Pierczyński
- Michał Pilipczuk
- Jakob Piribauer
- Vladimir Podolskii
- Gael Poette
- Filip Pokrývka
- Sheung-Hung Poon
- Alexandru Popa
- Tom Portoleau
- Aditya Prakash
- Marcin Przybyłko
- Giulia Punzi
- Gabriele Puppis
- David Purser
- Mikaël Rabie
- Jakub Radoszewski
- Ashutosh Rai
- Mahesh Rajasree
- Deepak Rajendraprasad
- Sergio Rajsbaum
- Michael Raskin
- Gaurav Rattan
- Rodrigo Raya
- Adele Rescigno
- Cristian Riveros
- David Roberson
- Dana Ron-Goldreich
- Ron Rosenthal
- Dino Rossegger
- Peter Rossmanith
- Jurriaan Rot
- Sanjukta Roy
- Andrew Ryzhikov
- Paweł Rzażewski
- Mathieu Sablik
- Lakshay Saggi
- Vibha Sahlot
- Abhishek Sahu
- Ville Salo
- Joan Andreu Sanchez
- Ignasi Sau
- Zdeněk Sawa
- Robert Scheffler
- Andre Schidler
- Anders Schlichtkrull
- Lena Schlipf
- Jason Schoeters
- Lutz Schröder
- Carsten Schuermann
- Tessa Seeger
- Kazuhisa Seto
- Martin P. Seybold
- Amatyia Sharma

- Alexander Shen
- Alexandra Silva
- Sunil Simon
- Kirill Simonov
- Henry Sinclair-Banks
- Piotr Skowron
- Fabio Somenzi
- Krzysztof Sornat
- Uéverton Souza
- Ramanujan M. Sridharan
- Ludwig Staiger
- Yannis Stamatiou
- Giannos Stamoulis
- Rafał Stefański
- Dario Stein
- Warut Suksompong
- Céline Swennenhuis
- Stanisław Szufa
- Prafullkumar Tale
- Navid Talebanfard
- Xizhi Tan
- Till Tantau
- Sébastien Tavenas
- Gabriele Tedeschi
- Lidia Tendera
- Pascal Tesson
- K. S. Thejaswini
- Leonidas Theocharous
- Simone Tini
- Jacobo Torán
- Noam Touitou
- Dekel Tsur
- Mihir Vahanwala
- Danny Vainstein
- Steffen van Bergerem
- Marie Van Den Bogaard
- Wim Van den Broeck
- Ivor van der Hoog
- Tom van der Zanden
- Sam van Gool
- Geert van Wordragen
- Pierre Vandenhove
- Anton Varonka
- Giovanna Varricchio
- Oleg Verbitsky
- Pavel Veselý
- Jonni Virtema
- Mikhail Volkov
- Mikhail Vyalyi
- Magnus Wahlström
- Kangning Wang
- Xiaofan Wang
- Yanhao Wang
- Dimitri Watel
- Simon Weber
- Mathias Weller
- Alexandra Wesolek
- Sebastian Wild
- Tobias Winkler
- Sarah Winter
- Michal Wlodarczyk
- Petra Wolf
- Guohua Wu
- Zhilin Wu
- Jules Wolms
- Mingji Xia
- Mingyu Xiao
- Chenyang Xu
- Zhiyang Xun
- Kevin Yeo
- Hantao Yu
- Chihao Zhang
- Fred Zhang
- Guochuan Zhang
- Hanwen Zhang
- Hengjie Zhang
- Jialin Zhang
- Kuize Zhang
- Roman Zorn

### Organizing Committee

- Hugo Gimbert, **co-Chair** (LaBRI, Université Bordeaux, CNRS, Bordeaux-INP)
- Sylvain Lombardy, **co-Chair** (LaBRI, Université Bordeaux, CNRS, Bordeaux-INP)
- Anca Muscholl (LaBRI, Université Bordeaux, CNRS, Bordeaux-INP)
- Vincent Penelle (LaBRI, Université Bordeaux, CNRS, Bordeaux-INP)



# Exploring the Space of Colourings with Kempe Changes

Marthe Bonamy  

CNRS, LaBRI, Université de Bordeaux, France

---

## Abstract

Kempe changes were introduced in 1879 in an attempt to prove the 4-colour theorem. They are a convenient if not crucial tool to prove various colouring theorems. Here, we consider how to navigate from a colouring to another through Kempe changes. When is it possible? How fast?

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

**Keywords and phrases** Graph theory, graph coloring, reconfiguration

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.1

**Category** Invited Talk

Kempe changes were introduced in 1879 in an attempt to prove the 4-colour theorem [4]. They are a convenient if not crucial tool to prove various colouring theorems, most notably the 4-colour theorem [1] (every planar graph is 4-colourable) and Vizing’s edge colouring theorem [7] (the edges of any graph can be partitioned into at most  $\Delta + 1$  matchings, where  $\Delta$  is the maximum degree of a vertex in the graph). Given a coloured graph, a Kempe change consists in considering two colours  $a$  and  $b$  and a vertex coloured  $a$ , then swapping colours  $a$  and  $b$  in the maximal  $(a, b)$ -coloured component containing the specified vertex. Here, we consider how to navigate from a colouring to another through a series of Kempe changes. When is it possible? How fast? A seminal conjecture of Vizing from 1965 [8] states that in any graph, from any edge-colouring we can reach an optimal one through a well-chosen series of Kempe changes. While this remained a major challenge for decades, being only proved for graphs with maximum degree 3 or 4 [5, 2], then last year for triangle-free graphs [3], Narboni recently provided a full proof of the conjecture [6]. This notably implies that given at least one more colour than the optimal number, one can navigate from any edge-colouring to any other. The extra colour is necessary. How these results extend to the context of multigraphs remains widely open.

We will also discuss the number of steps necessary to navigate from any vertex-colouring of a  $k$ -degenerate graph to any other, when the number of colours is sufficiently large compared to the degeneracy, as well as the Kempe equivalent of Hadwiger’s conjecture and whether Kempe changes can be useful in the context of graphs with a forbidden minor. If time permits, we will see how this tool can be used for efficient sampling of random colourings of a graph and for counting the number of distinct colourings.

---

## References

- 1 Kenneth Appel and Wolfgang Haken. The solution of the four-color-map problem. *Scientific American*, 237(4):108–121, 1977.
- 2 Armen S. Asratian and Carl Johan Casselgren. Solution of Vizing’s problem on interchanges for the case of graphs with maximum degree 4 and related results. *Journal of Graph Theory*, 82(4):350–373, 2016.
- 3 Marthe Bonamy, Oscar Defrain, Tereza Klimošová, Aurélie Lagoutte, and Jonathan Narboni. On vizing’s edge colouring question. *Journal of Combinatorial Theory, Series B*, 159:126–139, 2023.



© Marthe Bonamy;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 1; pp. 1:1–1:2

Leibniz International Proceedings in Informatics



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

## 1:2 Exploring the Space of Colourings with Kempe Changes

- 4 Alfred B. Kempe. On the geographical problem of the four colours. *American journal of mathematics*, 2(3):193–200, 1879.
- 5 Jessica McDonald, Bojan Mohar, and Diego Scheide. Kempe equivalence of edge-colorings in subcubic and subquartic graphs. *Journal of Graph theory*, 70(2):226–239, 2012.
- 6 Jonathan Narboni. Vizing’s edge-recoloring conjecture holds. *arXiv preprint arXiv:2302.12914*, 2023.
- 7 Vadim G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.
- 8 Vadim G. Vizing. Some unsolved problems in graph theory. *Russian Mathematical Surveys*, 23(6):125, 1968.

# Online Algorithms with Predictions

Joan Boyar   

University of Southern Denmark, Odense, Denmark

---

## Abstract

We give an introduction to online algorithms with predictions, from an algorithms researcher's perspective, concentrating on minimization problems.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online algorithms with predictions, online algorithms with advice, random order analysis

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.2

**Category** Invited Talk

**Funding** Supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B, and in part by the Innovation Fund Denmark, grant 9142-00001B, Digital Research Centre Denmark, project P40: Online Algorithms with Predictions.

## 1 Extended Abstract

We begin with an introduction to online algorithms with predictions, where the online algorithm is given additional information, some predictions, which should, presumably, improve its performance. The seminal papers in this area, by Lykouris and Vassilvitskii [9, 10] and Purohit, Svitkina and Kumar [11], appeared at conferences in 2018 and have since inspired many other researchers to work in this area. A current list of related publications can be found on a dedicated website [1], which listed 145 articles as of July 10, 2023.

Online algorithms are those that as input receive a sequence of requests, each of which must be handled by the algorithm making an irrevocable decision, before the next request arrives. The research area, online algorithms with predictions, is related to an older line of research within online algorithms, advice complexity [5, 7, 4, 6], where the online algorithm is given “advice” which is assumed to be correct. In contrast, in the area of online algorithms with predictions, the predictions given to these online algorithms may come from machine learning and generally contain errors. In both of these models, the online algorithm receives extra information and the performance of an algorithm is measured using the competitive ratio (asymptotically, the worst-case ratio over all possible input sequences of the cost obtained by the algorithm compared to the cost of OPT, the optimal offline algorithm, on the same input). In advice complexity, the goal is to achieve a good competitive ratio with as few bits of advice as possible. In contrast, the goal in online algorithms with predictions is to achieve a good competitive ratio, despite errors in the predictions.

The amount of error in the predictions,  $\hat{p}$ , for an input sequence  $I$  with correct values,  $p$ , is given by some error measure,  $\eta(I, \hat{p}, p)$ . This is typically normalized as  $\eta(I, \hat{p}, p) / \text{OPT}(I)$ , where  $\text{OPT}(I)$  is the cost achieved by an optimal offline algorithm on input sequence  $I$ . An online algorithm with predictions should have a competitive ratio that degrades gracefully with increasing error (smoothness), performing near optimally if there is no error (consistency), but not performing too poorly, even if the predictions are terrible (robustness).

Results for two different minimization problems are presented, demonstrating

- the relevance of advice complexity for the paging problem with predictions [2] and
- the relevance of random order analysis [8] for a problem where, according to competitive analysis, no algorithm can be better than the trivial Follow-the-Predictions algorithm [3].



© Joan Boyar;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 2; pp. 2:1–2:2

Leibniz International Proceedings in Informatics



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

---

**References**

---

- 1 Algorithms with predictions. <https://algorithms-with-predictions.github.io/>. Accessed: 2023-07-10.
- 2 Antonios Antoniadis, Joan Boyar, Marek Eliás, Lene M. Favrholdt, Ruben Hoeksma, Kim S. Larsen, Adam Polak, and Bertrand Simon. Paging with succinct predictions. *CoRR*, abs/2210.02775, 2022. To appear in ICML 2023. doi:10.48550/arXiv.2210.02775.
- 3 Magnus Berg, Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online minimum spanning trees with weight predictions. *CoRR*, abs/2302.12029, 2023. To appear in WADS 2023. doi:10.48550/arXiv.2302.12029.
- 4 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. Online algorithms with advice: The tape model. *Information and Computation*, 254:59–83, 2017.
- 5 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theor. Inf. Appl.*, 43(3):585–613, 2009.
- 6 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412:2642–2656, 2011.
- 7 Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *MFCS*, volume 6281 of *LNCS*, pages 24–36. Springer, 2010.
- 8 Claire Kenyon. Best-fit bin-packing with random order. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–364. SIAM, 1996.
- 9 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In Jennifer G. Dy and Andreas Krause, editors, *35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311, 2018.
- 10 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM*, 68(4):24:1–24:25, 2021.
- 11 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *31st Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 9684–9693. Curran Associates, Inc., 2018.

# Modern Parallel Algorithms

Artur Czumaj   

University of Warwick, Coventry, UK

---

## Abstract

Recent advances in the design of efficient parallel algorithms have been largely focusing on the nowadays classical model of parallel computing called *Massive Parallel Computation (MPC)*, which follows the framework of MapReduce systems. In this talk we will survey recent advances in the design of algorithms for graph problems for the MPC model and will mention some interesting open questions in this area.

**2012 ACM Subject Classification** Theory of computation → Massively parallel algorithms; Theory of computation → Distributed algorithms; Mathematics of computing → Graph algorithms

**Keywords and phrases** Distributed computing, parallel computing

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.3

**Category** Invited Talk

**Funding** *Artur Czumaj*: Research supported in part by the Centre for Discrete Mathematics and its Applications, by EPSRC award EP/V01305X/1, by a Weizmann-UK Making Connections Grant, by an IBM Award, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

## 1 Overview

Many modern computer applications require performing computations on massive amounts of data. Traditional models of computation, such as the RAM model or shared-memory parallel systems, are often inadequate for such computations, as the input do not fit into the available memory of even most advanced modern systems. The restrictions imposed by the limited memory in the available architectures and the requirement of fast processing of data has naturally led to the development of new models of parallel and distributed computation that are more suitable for processing massive amounts of data. On the basis of the successes of such massively parallel computation frameworks, such as MapReduce, Hadoop, Dryad, or Spark, Karloff, Suri, and Vassilvitskii (SODA 2010) introduced the *Massive Parallel Computation (MPC) model* that provides a clean abstraction of these frameworks and captures the modern needs of computation at a massive scale. After some later refinements, the MPC model has become the standard theoretical model of algorithmic study. At a very high-level, an MPC system consists of a collection of machines that can communicate with each other through indirect communication channels. The computation proceeds in synchronous rounds, where at each round the machines receive messages from other machines, perform arbitrarily complex local computations, and finally send appropriate messages to other machines so that the next round can start. The crucial factors in the analysis of algorithms in the MPC model are the number of rounds and the capacity of individual machines.

In the MPC model, there are  $m$  machines and each of them has  $s$  words of local space at its disposal. Initially, each machine receives its share of the input. For example, in the context of graph problems where the input is a collection  $V$  of nodes and  $E$  of edges, the input is arbitrarily distributed among the machines (and so  $s \cdot m \geq |V| + |E|$ ). The computation proceeds in synchronous *rounds* in which each machine processes its local data and performs an arbitrary complex local computation on its data. At the end of each round, machines exchange messages. Each message is sent only to a single machine specified by the machine



© Artur Czumaj;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 3; pp. 3:1–3:2

Leibniz International Proceedings in Informatics



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

## 3:2 Modern Parallel Algorithms

that is sending the message. All messages sent and received by each machine in each round, as well as the output have to fit into the machine's local space  $\mathfrak{s}$  (in particular, in a single round, any machine can send at most  $\mathfrak{s}$  messages and be the recipient of at most  $\mathfrak{s}$  messages).

It has been quickly observed that the central parameter of the MPC is its local space  $\mathfrak{s}$ . While originally the main research has been frequently focused on the case when  $\mathfrak{s}$  is almost as large as the input size, most recent study has been concentrated on the *low-space regime* when  $\mathfrak{s} = N^\phi$  for some  $\phi \in (0, 1)$ , often  $\phi$  being arbitrarily small.

The talk will survey this topic, focusing on graph problems for the low-space regime. We will discuss recent advances in the design of algorithms for graph problems for the MPC model for fundamental problems like connectivity and matching. We will also study the relation between the MPC model and some other fundamental models of parallel and distributing computations, including the classical PRAM model and the distributed LOCAL and Congested Clique models. We will also list some interesting open questions in this area.

# Algebraic Reasoning for (Un)Solvable Loops

Laura Kovács  

TU Wien, Austria

---

## Abstract

Loop invariants describe valid program properties that hold before and after every loop iteration. As such, loop invariants are the workhorses in formalizing loop semantics and automating the formal analysis and verification of programs with loops.

While automatically synthesizing loop invariants is, in general, an uncomputable problem, when considering only single-path loops with linear updates (linear loops), the strongest polynomial invariant is in fact computable [5, 9, 6, 3]. Yet, already for loops with “only” polynomial updates, computing the strongest invariant has been an open challenge since 2004 [8].

In this invited talk, we first present computability results on polynomial invariant synthesis for restricted polynomial loops, called *solvable* loops [11]. Key to solvable loops is that one can automatically compute invariants from closed-form solutions of algebraic recurrence equations that model the loop behaviour [6, 4]. We also establish a technique for invariant synthesis for classes of loops that are not solvable, termed *unsolvable* loops [1].

We next study the limits of computability in deriving the (strongest) polynomial invariants for *arbitrary* polynomial loops. We prove that computing the strongest polynomial invariant of arbitrary, single-path polynomial loops is very hard [10] – namely, it is at least as hard as the Skolem problem [2, 12], a prominent algebraic problem in the theory of linear recurrences. Going beyond single-path loops, we show that the strongest polynomial invariant is uncomputable already for multi-path polynomial loops with arbitrary quadratic polynomial updates [7].

**2012 ACM Subject Classification** Theory of computation → Invariants; Theory of computation → Program verification; Theory of computation → Algebraic semantics

**Keywords and phrases** Symbolic Computation, Formal Methods, Loop Analysis, Polynomial Invariants

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.4

**Category** Invited Talk

**Funding** We acknowledge funding from the European Research Council Consolidator Grant ARTIST 10100268, the Vienna Science and Technology Fund WWTF 10.47379/ICT19018 grant ProbInG, the Austrian Science Fund (FWF) project W1255-N23, the SecInt Doctoral College funded by TU Wien, and the European Union’s Horizon 2020 research and innovation programme LogiCS@TUWien under grant agreement No 101034440.

**Acknowledgements** This talk is based on joint works with a number of authors, including Daneshvar Amrollahi (TU Wien alumni), Ezio Bartocci (TU Wien), George Kenison (TU Wien), Marcel Moosbrugger (TU Wien), Julian Müllner (TU Wien), Miroslav Stankovic (TU Wien), and Anton Varonka (TU Wien).

---

## References

- 1 Daneshvar Amrollahi, Ezio Bartocci, George Kenison, Laura Kovács, Marcel Moosbrugger, and Miroslav Stankovic. Solving Invariant Generation for Unsolvable Loops. In *Proc. of SAS*, 2022. doi:10.1007/978-3-031-22308-2\_3.
- 2 Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. *Recurrence Sequences*. Mathematical Surveys and Monographs. American Mathematical Society, 2003.
- 3 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial Invariants for Affine Programs. In *Proc. of LICS*, 2018. doi:10.1145/3209108.3209142.



© Laura Kovács;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 4; pp. 4:1–4:2

Leibniz International Proceedings in Informatics



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

## 4:2 Algebraic Reasoning for (Un)Solvable Loops

- 4 Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. Automated Generation of Non-Linear Loop Invariants Utilizing Hypergeometric Sequences. In *Proc. of ISSAC*, 2017. doi:10.1145/3087604.3087623.
- 5 Michael Karr. Affine Relationships Among Variables of a Program. *Acta Inform.*, 1976. doi:10.1007/BF00268497.
- 6 Laura Kovács. Reasoning Algebraically About P-Solvable Loops. In *Proc. of TACAS*, 2008. doi:10.1007/978-3-540-78800-3\_18.
- 7 Laura Kovács and Anton Varonka. What Else is Undecidable About Loops? In *Proc. of RAMiCS*, 2023. doi:10.1007/978-3-031-28083-2\_11.
- 8 Markus Müller-Olm and Helmut Seidl. A Note on Karr’s Algorithm. In *Proc. of ICALP*, 2004. doi:10.1007/978-3-540-27836-8\_85.
- 9 Markus Müller-Olm and Helmut Seidl. Computing Polynomial Program Invariants. *Inf. Process. Lett.*, 2004. doi:10.1016/j.ipl.2004.05.004.
- 10 Julian Müllner. Exact Inference for Probabilistic Loops. Master’s thesis, TU Wien, 2023.
- 11 Enric Rodríguez-Carbonell and Deepak Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *Proc. of ISSAC*, 2004. doi:10.1145/1005285.1005324.
- 12 Terrence Tao. *Structure and Randomness*. American Mathematical Society, 2008.

# Sliding into the Future: Investigating Sliding Windows in Temporal Graphs

Nina Klobas  

Department of Computer Science, Durham University, UK

George B. Mertzios  

Department of Computer Science, Durham University, UK

Paul G. Spirakis  

Department of Computer Science, University of Liverpool, UK

---

## Abstract

Graphs are fundamental tools for modelling relations among objects in various scientific fields. However, traditional static graphs have limitations when it comes to capturing the dynamic nature of real-world systems. To overcome this limitation, *temporal graphs* have been introduced as a framework to model graphs that change over time. In temporal graphs the edges among vertices appear and disappear at specific time steps, reflecting the temporal dynamics of the observed system, which allows us to analyse time dependent patterns and processes. In this paper we focus on the research related to *sliding time windows* in temporal graphs. Sliding time windows offer a way to analyse specific time intervals within the lifespan of a temporal graph. By sliding the window along the timeline, we can examine the graph's characteristics and properties within different time periods.

This paper provides an overview of the research on sliding time windows in temporal graphs. Although progress has been made in this field, there are still many interesting questions and challenges to be explored. We discuss some of the open problems and highlight their potential for future research.

**2012 ACM Subject Classification** Mathematics of computing → Graph theory; General and reference → Surveys and overviews

**Keywords and phrases** Temporal Graphs, Sliding Time Windows

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.5

**Category** Invited Talk

**Funding** *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.

*Paul G. Spirakis*: Supported by the EPSRC grant EP/P02002X/1.

## 1 Introduction

Graphs are used to model (binary) relations among different objects. They consist of a set of vertices, where two of them are connected together with an edge. They have become a fundamental tool for modelling diverse systems and real-world problems, steaming through the wide range of scientific fields. Let us mention just a few of them. In Social sciences they can be used to model different interactions among people (for example friendships, communications, etc.), in Chemistry they can model chemical compounds where the vertices represent different atoms of the compound and edges correspond to the chemical bonds among them, in molecular Biology they can model physical interactions between proteins, gene co-expression or biochemical reactions, in Physics they model interactions among particles, where nodes represent interactions where particles are created or destroyed and edges are particles traveling between the interactions. Having such a varied application and use, it is not surprising that the graph theory has been the subject of extensive research over the past centuries.



© Nina Klobas, George B. Mertzios, and Paul G. Spirakis;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 5; pp. 5:1–5:12

Leibniz International Proceedings in Informatics



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

When studying real-life problems, it becomes evident that this ‘simple’ graph model is often insufficient. Many problems exhibit dynamic behavior, where the connections or interactions among their vertices change over time. For instance, in transportation networks, specific roads may be closed during certain intervals. In social networks, individuals may only interact at particular times of the day or month. Similarly, in information and communication networks, information or messages are transmitted from a source to a recipient through a set of connections at specific times. These graph models share a common attribute: their underlying graph topology or network structure undergoes discrete changes over time. This observation naturally gives rise to the concept of temporal graphs, which provide a straightforward and intuitive model for representing graphs that change over time, called *temporal graphs*.

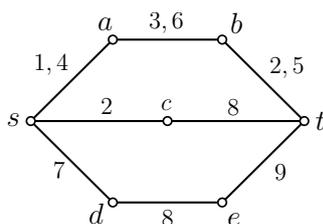
► **Definition 1** (temporal graph [24]). *A temporal graph  $\mathcal{G}$  is a pair  $(G, \lambda)$ , where  $G = (V, E)$  is an underlying (static) graph and  $\lambda : E \rightarrow 2^{\mathbb{N}}$  is a time labeling function which assigns to every edge of  $G$  a set of discrete time labels.*

Due to their relevance and applicability in many areas, temporal graphs have been studied from various perspectives and under different names such as *dynamic* [9,19], *evolving* [7,12,16], *time-varying* [1,17,36], and *graphs over time* [27].

In most applications of temporal graphs, information can naturally only move along edges in a way that respects the ordering of their timestamps (i.e. time labels), that is, information can only flow along sequences of edges whose time labels are increasing (or non-decreasing). Motivated by this fact, most studies on temporal graphs have focused on “path-related” problems, such as e.g. temporal analogues of distance, diameter, reachability, exploration, and centrality [2,3,10,14,15,22,25,26,30,34,40]. In these problems, the most fundamental notion is that of a *temporal path* from a vertex  $u$  to a vertex  $v$ , which is a path from  $u$  to  $v$  such that the time labels of the edges are increasing (or at least non-decreasing) in the direction from  $u$  to  $v$ . To complement this direction, several attempts have been recently made to define meaningful “non-path” temporal graph problems which appropriately model specific applications. Some examples include temporal cliques, cluster editing, temporal vertex cover, temporal graph coloring, temporally transitive orientations of temporal graphs [4,6,11,18,21,23,32,33,37,39].

One of the main goals in temporal graphs’ research is to lift (algorithmic) graph theory models and results to a temporal/dynamic domain, in order to model natural, real world situations which are subject to discrete changes over time. The main challenge in this front is to find appropriate natural extensions and definitions of such problems. For instance, in static graphs, a *shortest path* between two vertices is a path connecting these two vertices with the smallest number of edges. On the other hand, in temporal graphs, there are at least three, equally natural, different analogues of a shortest path. First, a *shortest* temporal path from  $u$  to  $v$  is one that contains the smallest number of edges. Second, a *foremost* temporal path from  $u$  to  $v$  is one that arrives at  $v$  with the smallest time-stamp. Third, a *fastest* temporal path from  $u$  to  $v$  is one that has the smallest duration. These three types of temporal paths are illustrated in Figure 1.

What is common to most of the path-related problems is that their extension from static to temporal graphs often follows easily and quite naturally from their static counterparts. For example, requiring a graph to be (temporally) connected results in requiring the existence of a (temporal) path among each pair of vertices. In the case of non-path related problems, the exact definition and its application is not so straightforward. Let us consider the case of cliques. Defining cliques in a temporal graph as the set of vertices that interact at least once in the lifetime of the graph would be a bit counter intuitive, as two vertices may just interact



■ **Figure 1** In this temporal graph, the shortest path from  $s$  to  $t$  is  $(s, c, t)$  as it contains two edges; the foremost temporal path is  $(s, a, b, t)$  as it arrives at time 5; the fastest temporal path is  $(s, d, e, t)$  as it has duration  $9 - 7 + 1 = 3$ .

at the first time step and never again. To help with this problem, Viard et al. [37] introduced the idea of the *sliding time window* of some size  $\Delta$ , where they define a temporal clique as a set of vertices where in all  $\Delta$  consecutive time steps each pair of vertices interacts at least once. There is a natural motivation for this problem, namely to be able to find the contact patterns among high-school students. Following the idea of Viard et al. [37], many other problems on temporal graph were defined using sliding time windows. In this paper we present an overview of works on sliding windows in temporal graphs and at the end provide some open problems and further ideas with potential research topics.

## 2 Preliminaries and Notations

In the literature there are many (slightly) different notations and terminologies used for certain structures in temporal graph. For the purpose of this paper, we fix the following notation and definitions.

Given a (static) graph  $G = (V, E)$  with vertices in  $V$  and edges in  $E$ , an edge between two vertices  $u$  and  $v$  is denoted by  $uv$ , and in this case  $u$  and  $v$  are said to be *adjacent* in  $G$ . For every  $i, j \in \mathbb{N}$ , where  $i \leq j$ , we let  $[i, j] = \{i, i + 1, \dots, j\}$  and  $[j] = [1, j]$ . Throughout the paper we consider temporal graphs whose underlying graphs are finite and whose time labeling functions only map to finite sets. This implies that there is some  $t \in \mathbb{N}$  such that, for every  $t' > t$ , no edge of  $G$  is active at  $t'$  in  $(G, \lambda)$ . We denote the smallest such  $t$  by  $T$ , i.e.,  $T = \max\{t \in \lambda(e) \mid e \in E\}$ , and call  $T$  the *lifetime* of  $(G, \lambda)$ . Unless otherwise specified,  $n$  denotes the number of vertices in the underlying graph  $G$ , and  $T$  denotes the lifetime of the temporal graph  $\mathcal{G}$ . We refer to each integer  $t \in [T]$  as a *time step* of  $(G, \lambda)$ . The *instance* (or *snapshot*) of  $(G, \lambda)$  at time  $t$  is the static graph  $G_t = (V, E_t)$ , where  $E_t = \{e \in E : t \in \lambda(e)\}$ . Note that the *size* of a temporal graph  $\mathcal{G}$  is  $|\mathcal{G}| := |V| + \sum_{t=1}^T |E_t|$ .

For every  $t = 1, \dots, T - \Delta + 1$ , let  $W_t = [t, t + \Delta - 1]$  be the  $\Delta$ -*time window* that starts at time  $t$ . For every  $v \in V$  and every time step  $t$ , we denote the *appearance of vertex  $v$  at time  $t$*  by the pair  $(v, t)$  and the *edge appearance* (or *time-edge*) of  $e$  at time  $t$  by  $(e, t)$ . For  $t \in \lambda(e)$  we also say that  $e$  is *active* at time  $t$  in  $(G, \lambda)$ . That is, for every edge  $e \in E$ ,  $\lambda(e)$  denotes the set of time steps at which  $e$  is *active*.

A *temporal vertex subset* of  $(G, \lambda)$  is a set of vertex appearances in  $(G, \lambda)$ , i.e. a set of the form  $S \subseteq \{(v, t) \mid v \in V, t \in [T]\}$ . For a temporal vertex subset  $S$  and some  $\Delta$ -time window  $W_i$  within the lifetime  $T$  of  $(G, \lambda)$ , we denote by  $S[W_i] = \{(v, t) \in S \mid t \in W_i\}$  the subset of all vertex appearances in  $S$  in the  $\Delta$ -time window  $W_i$ . For a  $\Delta$ -time window  $W_i$  within the lifetime of a temporal graph  $(G, \lambda)$ , we denote by  $E[W_i] = \{e \in E \mid \lambda(e) \cap W_i \neq \emptyset\}$  the set of all edges which appear at some time step within  $W_i$ .

### 3 Known Results on Sliding Windows

In this section we present some of the known results on temporal graphs using sliding windows. As we discussed, the aim is to find a suitable definition for well motivated graph problems, that take in consideration also the changes that appear over time.

#### 3.1 Temporal Cliques

In a (static) graph  $G = (V, E)$ , a clique  $C \subseteq V$  is a collection of vertices, where every two of them are connected. We say that a clique  $C$  is maximal, if there exists no other vertex in  $V \setminus C$  that is connected to all of the vertices in  $C$ . There are many applications of (maximal) cliques for modeling real-world problems. For example, in their work Creamer et al. [13] calculate hierarchical structures in complex (communication) networks using cliques, and in [35] Samudrala and Moulton use cliques in the context of protein structure modeling.

Viard et al. [37] extended the notion of cliques to temporal graphs. Their work was motivated by the contact patterns among French high-school students. They studied the dataset with real-world contacts between individuals, captured with sensors. Where an edge  $e$  at time  $t$  was formed between two subjects if they were close enough to each other at time  $t$  for the detection to happen. The aim is to determine groups of students that were interacting more often. The obstacle in this case is how to naturally define such groups. If two students interacted only once and then never again, their interaction should not be considered as “valuable” as in the case when students interact more often, over certain period of time. With this in mind, the authors present the following natural definition of a  $\Delta$ -clique.

► **Definition 2.** *A  $\Delta$ -clique  $C$  in a temporal graph  $\mathcal{G} = (G, \lambda)$  with a life-time  $T$ , is a pair  $(X, I)$ , where  $X$  is a subset of vertices of  $G$  and  $I \subseteq [T]$ , such that for every two vertices  $u, v \in X$  there is a time-edge  $(uv, t)$  in  $\mathcal{G}$  in every  $\Delta$ -time window  $W_i \in I$ .*

Intuitively, among each pair of vertices in  $X$  there is a time-edge every  $\Delta$  time steps in the time interval  $I$ . The significance of the parameter  $\Delta$  is that it measures the level of interaction in  $\Delta$ -cliques. A small value of  $\Delta$  means that the interaction among vertices has to occur more often compared to the case of large  $\Delta$  values. The selection of  $\Delta$  depends on the data set and the purpose of the analysis.

The authors provide an algorithm that in  $O(2^n n^2 \bar{m}^3 + 2^n n^3 \bar{m}^2)$  time computes all maximal  $\Delta$ -cliques of a temporal graph  $(G, \lambda)$ , where  $n = V(G)$  and  $\bar{m} = \sum_{e \in E(G)} \lambda(e)$ . This result was further improved by Himmel et. al. [23] by providing an adaptation of the Bron-Kerbosch algorithm for enumerating maximal cliques, where they improve the running time to  $O(2^n T m)$ , where  $m = |E(G)|$ .

Cliques may not be always practical for modelling real-world situations as they can be too restrictive, for example some edges may not exist due to measurement errors or other reasons specific to the application. To overcome this issue, various relaxations of the clique concept have been developed. One popular approach is the use of  $k$ -plexes, a degree-based relaxation of cliques that requires every vertex to be connected to all but at most  $k - 1$  vertices in the  $k$ -plex, excluding itself. Extending this idea to temporal graphs, Bentert et al. [6] introduce the study of  $\Delta - k$ -plexes, where they relax the condition of  $\Delta$ -clique by allowing each vertex to have up to  $k - 1$  missing connections to other vertices in each  $\Delta$  consecutive time steps. They adapt the algorithm for  $\Delta$ -cliques to enumerate them, and provide some heuristic speed-up techniques that are useful when dealing with practical scenarios.

### 3.2 Temporal Vertex Cover

The vertex cover problem on a static graph  $G$  asks for a set of vertices  $S$  in  $V(G)$ , of a minimum size, such that each edge  $e \in E(G)$  has at least one endpoint in  $S$  (i. e., is covered by at least one vertex in the vertex cover). To extend the idea to temporal graphs one needs to first find a relevant and well motivated definition. For example, requiring that each edge is covered whenever it appears (i. e., there is a vertex cover in every snapshot of the temporal graph), may be a bit too restrictive. A well known motivation behind the vertex cover on static graphs is a problem of placing security guards throughout the airport, where corridors represent edges and two corridors meet in a vertex. Then a vertex cover is a collection of corridor intersections, where we place security guards such that the airport is fully observed by the security. Suppose now that during the day, for some reason, certain corridors are not in use (some gates may be open only during specific times). And suppose now also that a criminal needs a specific amount of time, without any supervision, to execute an illegal activity. Now, to prevent all such acts, we do not need to fully monitor each sector of the airport all the time, but we just have to make sure we check each part often enough. With this in mind, Akrida et al. [4] introduced the notion of sliding window temporal vertex cover.

► **Definition 3.** *A  $\Delta$ -sliding window temporal vertex cover  $S \subseteq V(G) \times [T]$  (or  $\Delta$ -TVC for short) in a temporal graph  $(G, \lambda)$ , with a lifetime  $T$ , is a collection of vertex appearances, such that each edge  $e \in E(G)$  is covered in every  $\Delta$ -time window  $W_i \subseteq [T]$ , if it appears.*

When determining  $\Delta$ -TVC of a given temporal graph, one wants to always find the one of minimum size. In their work Akrida et al. [4] first prove that a relaxed version of the problem (where  $\Delta = T$ , i. e., each edge has to be covered at least once in the whole lifetime  $T$  of the graph) is NP-hard already for the temporal graphs where the underlying graph is a star. For this sub-problem they prove also that the optimal solution cannot be obtained in  $O(2^{\epsilon T})$  time (for some small  $\epsilon$ ), assuming the Strong Exponential Time Hypothesis (SETH), as well as that it does not admit a Polynomial-Time Approximation Scheme (PTAS). For the general problem they provide an exact dynamic algorithm running in  $O(T\Delta(n+m) \cdot 2^{n\Delta(\Delta+1)})$  time on arbitrary temporal graphs, which cannot admit much more improvements (as it is almost at the lower complexity bound). They complement this result by providing an algorithm that, for graphs where each snapshot has a vertex cover number bounded by  $k$ , runs in an FPT time, when parameterized by  $\Delta$ . They investigate also the problem's approximability and prove that  $\Delta$ -TVC does not admit a PTAS, even when  $\Delta = 2$ , maximum degree of the underlying graph is 3 and every connected component of each snapshot is of size at most 7. In addition, they augment this result by providing approximation algorithms with ratios (i)  $\ln n + \ln \Delta + \frac{1}{2}$ , (ii)  $2k$ , where  $k$  is the maximum number of appearances of an edge in a sliding window, (iii)  $d$ , where  $d$  is the maximum vertex degree in every snapshot.

The study of  $\Delta$ -TVC problem was then further extended by Hamm et. al. [21]. The researchers studied the  $\Delta$ -sliding window vertex cover problem on sparse temporal graphs. They proved that the problem is NP-hard when  $\Delta \geq 2$  and the underlying graph  $G$  of the temporal graph  $(G, \lambda)$  is a path or a cycle. On the other hand, they developed a polynomial-time algorithm for solving  $T$ -TVC on paths and cycles, where  $T$  is the lifetime of the temporal graph. This raises the interesting question of whether there exists a boundary value for  $\Delta$  that distinguishes between the tractable and intractable categories on paths, thus determining the complete dichotomy of the problem. Moreover, for any  $\Delta \geq 2$  they augmented these results with a PTAS for  $\Delta$ -TVC on paths and cycles, which complements the hardness result. In addition, the authors presented three algorithms to counter the

hardness of the  $\Delta$ -TVC problem for arbitrary (non-restricted) temporal graphs. The first algorithm is an exact algorithm for  $\Delta$ -TVC with an exponential running time dependency on the number of edges in the underlying graph. Using this algorithm, they developed a polynomial-time  $(d - 1)$ -approximation algorithm for any  $d \geq 3$ , where  $d$  is the maximum vertex degree in any time step (which improved on  $d$ -approximation algorithm from Akrida et al.). Finally, the authors presented a simple fixed-parameter tractable algorithm with respect to the size of an optimum solution.

### 3.3 Temporal Coloring

In a static graph  $G$  a coloring problem asks for a minimum number of colors associated to vertices, such that two endpoints of each edge are not assigned the same color. A classical motivation behind this problem is allocating radio frequencies to radio towers at specific locations. Here the idea is to allocate different frequencies to towers that are located close enough to cause an overlap in transmission. In this case each tower is represented as a node of the graph, where two of them are connected if the towers are positioned so close that they interfere with each other, and each frequency represents a different color. Now, coloring the graph properly results in an assignment of frequencies, that causes no interference.

Let us consider a bit more evolved scenario, where instead of static radio towers, we observe mobile agents. Here every agent broadcasts information over a specific communication channel while it listens on all others. Therefore, when two agents are in close proximity, they exchange information only if they broadcast on different channels. We assume that agents can switch channels at any time. To ensure maximum information exchange, it is essential to find a schedule of assigning broadcasting channels to the agents over time that minimizes the number of required channels. This should allow each pair of agents to communicate at least once within every small time window when they are close to each other.

Following this motivation Mertzios et al. [33] introduce the study of temporal coloring using sliding windows. Where one wants to determine the coloring of vertex appearances, using the smallest possible number of colors, such that each edge is properly colored (incident vertices are of different color) at least once in every  $\Delta$  consecutive time steps, if the edge appears. For a formal definition see the following.

► **Definition 4.** *A  $\Delta$ -sliding window temporal coloring (or  $\Delta$ -TC for short) in a temporal graph  $(G, \lambda)$ , with a lifetime  $T$ , is a function  $\phi : V(G) \times [T] \rightarrow \mathbb{N}$ , that assigns one color  $\phi(v, t)$  to each vertex appearance, such that for every  $\Delta$ -time window  $W_i \subseteq [T]$ , and every edge  $e \in E[W_i]$  there is at least one time step  $t \in W_i$ , where  $e$  appears and its two endpoints  $u, v$  are colored using different colors, i. e.,  $(e, t)$  is a time-edge in  $(G, \lambda)$  and  $\phi(u, t) \neq \phi(v, t)$ .*

Mertzios et al. [32] start by studying a subcase of the problem, when  $\Delta = T$ . In this case the objective is to ensure that each edge is properly colored (its endpoints are of different color) at least once in the whole life time  $T$  of the temporal graph. Surprisingly, even the restricted subcase turns out to be NP-hard, already when one is asking if 2 colors are enough to color it properly. This presents a stark contrast to the static case, where identifying if a graph is 2-colorable (bipartite) can be accomplished in linear time. On the positive side they show that this subcase admits a polynomial kernel, when parameterized by the number of vertices in the input temporal graph. For the general case they prove that the problem is NP-hard, and provide two algorithms for it. One is an exponential-time algorithm, that asymptotically matches the running time lower bound (assuming ETH), and the second one is a linear time FPT algorithm, with respect to the number  $n$  of vertices.

In addition to the above mentioned work, some other variations of coloring temporal graphs have been explored (e.g. example [18, 29, 39]), however these studies do not use the approach with sliding windows.

### 3.4 Temporal Matching

Given a static graph  $G$  the problem of (maximum) matching asks for a (maximum) set of pairwise independent edges, that is, edges that share no endpoints. This problem has numerous applications in fields such as scheduling and planning, chemistry modeling, job allocation, and more. Once the time-dimension is added to the graph model, there can be different ways to carry over the definition to temporal graphs.

Following the idea of a sliding time window, Mertzios et al. [31] introduced the problem of  $\Delta$ -MAXIMUM TEMPORAL MATCHING ( $\Delta$ -TM), where one wants to determine a maximum set of time-edges that are pairwise  $\Delta$ -independent. Two time-edges  $(e, t), (f, t')$  are  $\Delta$ -independent if (i)  $e \cap f = \emptyset$ , or (ii)  $e \cap f \neq \emptyset$  and  $|t - t'| \geq \Delta$ . In other words, for any feasible solution of  $\Delta$ -TM, it is not possible to match a vertex more than once within any time interval of duration  $\Delta$ . This condition can represent scenarios where a short “recovery” period is needed for every vertex that participates in the matching, such as a brief period of rest after engaging in an energy-demanding activity.

In contrast to the Edmonds’ polynomial-time algorithm for finding a maximum matching in static graphs, Mertzios et al. [31] prove that  $\Delta$ -TM does not even admit an approximation algorithm, meaning it is APX hard, already in the case when  $\Delta = 2$  and the lifetime  $T$  of the temporal graph is 3. In addition, they show that the problem remains NP-hard even if the underlying graph, of the input temporal graph, is just a path. On the positive side, they provide an approximation algorithm for any constant  $\Delta$ , which achieves an approximation ratio of  $\frac{1}{2} + \epsilon$ , where  $\epsilon = \frac{1}{2(2\Delta-1)}$ . Besides that, they show that a problem admits two FPT algorithms, one when it is parameterized by the solution size, and the second one, when it is parameterized by the combined parameter  $\Delta$  and the size of a maximum matching of the underlying graph.

It is worth mentioning that another related variant of MAXIMUM TEMPORAL MATCHING has been studied (see Baste et al. [5]). In this model the authors do not use the  $\Delta$ -time windows, but instead require an edge to appear at least  $\Delta$  consecutive time steps, in order to be eligible for a matching. A temporal matching then consists of independent edge time-blocks of length at least  $\Delta$ .

## 4 Further Work

In the previous section we presented some already completed works on temporal graphs, that use the idea of sliding time windows. In this section we focus on problems that, to the best of our knowledge, have not yet been investigated using the sliding windows, and give rise to some interesting research questions.

### 4.1 Dominating Set

In a static graph  $G$ , a dominating set is a subset of vertices  $D \subseteq V(G)$ , such that each vertex  $V(G)$  is either in  $D$  or has a neighbor in  $D$ . The *Dominating set problem* asks for a dominating set of  $G$  of minimum size. One of the applications of the dominating set is in routing protocols for ad hoc wireless networks. The fundamental concept behind this approach involves identifying a dominating set within a network of devices and using these

dominating nodes for message routing. More specifically, when a user  $u$  wants to transmit a message to a user  $v$ , the routing process consists of determining the shortest path between the dominating neighbors of user  $u$  and user  $v$ . By ensuring that all devices admit at least one dominating neighbor, this method guarantees the delivery of messages.

In the world where these agents become mobile (i. e., they travel around the space), one can model this problem using temporal graph, where the aim is to find a *temporal dominating set*. Similarly as in other cases, we do not necessarily want to find a dominating set in every time step (as this would be too costly), therefore an approach with sliding time windows would be of use. We propose the following definition.

► **Definition 5.** A  $\Delta$ -sliding window temporal dominating set ( $\Delta$ -SWDS) is a subset of vertex appearances  $D \subseteq V(G) \times [T]$ , of a temporal graph  $(G, \lambda)$ , with the lifetime  $T$ , such that for any vertex appearance  $(v, t)$  the following holds:

1.  $(v, t') \in D$ , where  $|t - t'| \leq \Delta$  or
2.  $(u, t') \in D$ , where  $u$  is a neighbor of  $v$  in  $G$  and  $|t - t'| \leq \Delta$ .

Intuitively, any vertex of the underlying temporal graph is at any time step  $t$  either at most  $\Delta$  time-units away from being in  $D$ , or it has a neighbor that is at most  $\Delta$  time-units away from being in  $D$ . Since the Dominating set problem is already NP-hard on static graphs, it remains hard also for temporal graphs. So the interesting research question for the  $\Delta$ -SWDS would be if there exist any exact algorithms for it, i. e., some FPT algorithms, or maybe some approximation algorithms.

It is important to mention that there already exist some variations of the dominating set problems on temporal graphs. Casteigts and Flocchini [8] propose three different definitions of dominating sets on temporal graphs, namely temporal dominating set, evolving dominating set and permanent dominating set. In the temporal and evolving dominating set, one wants to determine the smallest set of vertices  $D$ , such that, in the temporal case, each vertex is dominated in at least one time step, and in the evolving case, each vertex is dominated in every time step. While the evolving dominating set  $D$  consists of vertex appearances, such that all vertex appearances are dominated in each time step. More specifically, in the first two cases, once a vertex is selected to be in  $D$  it is in  $D$  for all lifetime of the graph, while in the last case one vertex can be in  $D$  only at specific times. Some research has been done for aforementioned problems. For interested readers, we recommend exploring the following works [20, 28, 38], among others.

## 4.2 Edge Cover

The *minimum edge covering problem* on a static graph  $G$  asks for a minimum set  $E_C \subseteq E(G)$  of edges such that every vertex in  $V(G)$  is incident to at least one edge in  $E_C$ . Calculating a minimum edge cover can be done in polynomial time, by finding a maximum matching and then extending it greedily until all vertices are covered.

For the version of the edge covering problem on temporal graphs we propose the following definition.

► **Definition 6.** A  $\Delta$ -sliding window temporal edge covering ( $\Delta$ -SWECC) is a subset of edge appearances  $EC \subseteq E(G) \times [T]$ , of a temporal graph  $(G, \lambda)$ , with the lifetime  $T$ , such that every vertex appearance  $(v, t)$  is incident to at least one time-edge from the selected set  $EC \subseteq E \times T$ , in every time window  $t \in W_t$ .

Since many problems become significantly more challenging when dealing with temporal graphs, it would be really interesting to explore whether the same holds true for the  $\Delta$ -SWECC problem. Applying the exact approach used for static graphs may not yield direct results, as

it requires to first find a (suitable definition of a) temporal maximum matching. It is worth noting that in Section 3.4 we presented a  $\Delta$ -MAXIMUM TEMPORAL MATCHING, which turns out to be NP-hard.

### 4.3 Periodic connectivity

We say that a temporal graph  $(G, \lambda)$  is *temporally connected* if there exists a temporal path among each pair of vertices. Some results regarding the connectivity of temporal graphs have already been established, for example [3, 26, 30]. However, what if we introduce additional constraints and require that each vertex can reach any other vertex within every  $\Delta$  time-window? In such cases, we refer to the temporal graph  $(G, \lambda)$  as being  $\Delta$ -*temporally connected*. It would be interesting to study, for example, what is the minimum number of labels needed to label a given graph  $G$  in such a way that ensures  $\Delta$ -temporal connectivity of  $(G, \lambda)$ ? We can further restrict this problem by allowing only limited number  $k$  of labels to be added per each edge.

### 4.4 (Temporal) Graph Classes

Based on the properties of the studied graphs, we can assign them into different graph classes. For instance there are graphs that are  $k$ -colorable (can be properly colored using  $k$  colors),  $k$ -regular (each vertex is of degree  $k$ ), or planar (can be drawn on a plane without any edges crossing), among others.

To extend the concept of graph classes to the temporal setting with sliding windows, we propose introducing temporal graph classes. One such class could be the  $\Delta$  *sliding window  $k$ -colorable temporal graphs*, which refers to temporal graphs that can be temporally colored using  $k$  colors. Another class would be the  $\Delta$  *sliding window  $k$ -regular temporal graphs*, where each vertex admits exactly  $k$  different neighbors in every  $\Delta$  time-window, or perhaps each vertex  $v$  admits exactly  $k$  different neighbors in a time step  $t' \in W_t$  for every time window  $W_t$ . Similarly, we can define  $\Delta$  *sliding window planar temporal graphs* as temporal graphs that are planar in some  $t' \in W_t$  for every time-window  $W_t$ . Further refinement of these classes is possible by imposing additional restrictions. For example, we can consider temporal graphs that are *3-colorable in every 5-time window*. In such graphs, every vertex appearance  $(v, t')$  is assigned one of three colors, ensuring that within each sliding window  $W_t$  of size 5, there is at least one time step where the edge  $e$ , that appears in  $W_t$  is properly colored.

Overall, these extensions allow for the classification of temporal graphs based on their temporal characteristics, enabling the exploration of various graph classes in the context of sliding windows.

## 5 Conclusion

The study of temporal graphs has emerged as an important area of research with significant implications for understanding and analyzing dynamic systems. In this paper, we have presented a short overview of the works on sliding windows in temporal graphs. The concept of a sliding time window allows us to focus on specific temporal intervals within the lifetime of a temporal graph, providing valuable insights into the changing behavior and patterns of interactions. Given that this research field is fairly young, there are still many intriguing questions and challenges to be addressed. We presented some of them here and hope that this work inspires further exploration and investigation into these intriguing problems.

---

References

---

- 1 Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 29–41, 2014.
- 2 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.
- 3 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017.
- 4 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020.
- 5 Julien Baste, Binh-Minh Bui-Xuan, and Antoine Roux. Temporal matching. *Theor. Comput. Sci.*, 806:184–196, 2020.
- 6 Matthias Bentert, Anne-Sophie Himmel, Hendrik Molter, Marco Morik, Rolf Niedermeier, and René Saitenmacher. Listing all maximal  $k$ -plexes in temporal graphs. In *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 41–46, 2018.
- 7 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.
- 8 Arnaud Casteigts and Paola Flocchini. Deterministic algorithms in dynamic networks: Problems, analysis, and algorithmic tools. Technical report, Defence R&D Canada, 2013.
- 9 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.
- 10 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.
- 11 Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster editing in multi-layer and temporal graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, volume 123, pages 24:1–24:13, 2018.
- 12 Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time of edge-markovian evolving graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1694–1712, 2010.
- 13 Germán Creamer, Ryan Rowe, Shlomo Hershkop, and Salvatore J. Stolfo. Segmentation and automated social hierarchy detection through email network analysis. In *Advances in Web Mining and Web Usage Analysis, 9th International Workshop on Knowledge Discovery on the Web, WebKDD 2007*, Lecture Notes in Computer Science, pages 40–58, 2007.
- 14 Jessica A. Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.
- 15 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021.
- 16 Afonso Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.
- 17 Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.
- 18 Subhankar Ghosal and Sasthi C Ghosh. Channel assignment in mobile networks based on geometric prediction and random coloring. In *Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN)*, pages 237–240, 2015.

- 19 George Giakkoupis, Thomas Sauerwald, and Alexandre Stauffer. Randomized rumor spreading in dynamic graphs. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 495–507, 2014.
- 20 Leonidas J. Guibas, Nikola Milosavljevic, and Arik Motskin. Connected dominating sets on dynamic geometric graphs. *Comput. Geom.*, 46(2):160–172, 2013.
- 21 Thekla Hamm, Nina Klobas, George B. Mertzios, and Paul G. Spirakis. The complexity of temporal vertex cover in small-degree graphs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, February 22 - March 1, 2022*, pages 10193–10201, 2022.
- 22 Klaus Heeger, Danny Hermelin, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling on a single machine. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 11818–11825. AAAI Press, 2021.
- 23 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.
- 24 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.
- 25 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: temporally disjoint paths. *Auton. Agents Multi Agent Syst.*, 37(1), 2023.
- 26 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The complexity of computing optimum labelings for temporal connectivity. In *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241, pages 62:1–62:15, 2022.
- 27 Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.
- 28 Subhrangsu Mandal and Arobinda Gupta. Approximation algorithms for permanent dominating set problem on dynamic networks. In *Distributed Computing and Internet Technology - 14th International Conference, ICDCIT 2018, Bhubaneswar, India, January 11-13, 2018, Proceedings*, volume 10722 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2018.
- 29 Andrea Marino and Ana Silva. Coloring temporal graphs. *Journal of Computer and System Sciences*, 123:171–185, 2022.
- 30 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.
- 31 George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154, pages 27:1–27:14, 2020.
- 32 George B. Mertzios, Hendrik Molter, Malte Renken, Paul G. Spirakis, and Philipp Zschoche. The complexity of transitively orienting temporal graphs. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 75:1–75:18, 2021.
- 33 George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. Sliding window temporal graph coloring. *Journal of Computer and System Sciences*, 120:97–115, 2021.
- 34 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.
- 35 Ram Samudrala and John Moulton. A graph-theoretic algorithm for comparative modeling of protein structure. Edited by f. cohen. *Journal of Molecular Biology*, 279(1):287–302, 1998.
- 36 John Kit Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *Computer Communication Review*, 40(1):118–124, 2010.

## 5:12 Sliding into the Future: Investigating Sliding Windows in Temporal Graphs

- 37 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.
- 38 John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *The 18th Annual International Conference on Mobile Computing and Networking, Mobicom'12, Istanbul, Turkey, August 22-26, 2012*, pages 377–388. ACM, 2012.
- 39 Feng Yu, Amotz Bar-Noy, Prithwish Basu, and Ram Ramanathan. Algorithms for channel assignment in mobile wireless networks using temporal coloring. In *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems (MSWiM)*, pages 49–58, 2013.
- 40 Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.

# Roman Census: Enumerating and Counting Roman Dominating Functions on Graph Classes

Faisal N. Abu-Khzam  

Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

Henning Fernau   

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Kevin Mann  

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

---

## Abstract

---

The concept of Roman domination has recently been studied concerning enumerating and counting in F. N. Abu-Khzam et al. (WG 2022). More technically speaking, a function that assigns 0, 1, 2 to the vertices of an undirected graph is called a *Roman dominating function* if each vertex assigned zero has a neighbor assigned two. Such a function is called *minimal* if decreasing any assignment to any vertex would yield a function that is no longer a Roman dominating function. It has been shown that minimal Roman dominating functions can be enumerated with polynomial delay, i.e., between any two outputs of a solution, no more than polynomial time will elapse. This contrasts what is known about minimal dominating sets, where the question whether or not these can be enumerated with polynomial delay is open for more than 40 years. This makes the concept of Roman domination rather special and interesting among the many variants of domination problems studied in the literature, as it has been shown for several of these variants that the question of enumerating minimal solutions is tightly linked to that of enumerating minimal dominating sets, see M. Kanté et al. in SIAM J. Disc. Math., 2014. The running time of the mentioned enumeration algorithm for minimal Roman dominating functions (Abu-Khzam et al., WG 2022) could be estimated as  $\mathcal{O}(1.9332^n)$  on general graphs of order  $n$ . Here, we focus on special graph classes, as has been also done for enumerating minimal dominating sets before. More specifically, for chordal graphs, we present an enumeration algorithm running in time  $\mathcal{O}(1.8940^n)$ . It is unknown if this gives a tight bound on the maximum number of minimal Roman dominating functions in chordal graphs. For interval graphs, we can lower this time bound further to  $\mathcal{O}(1.7321^n)$ , which also matches the known lower bound concerning the maximum number of minimal Roman dominating functions. We can also provide a matching lower and upper bound for forests, which is (incidentally) the same, namely  $\mathcal{O}^*(\sqrt{3}^n)$ . Furthermore, we present an optimal enumeration algorithm running in time  $\mathcal{O}^*(\sqrt[3]{3}^n)$  for split graphs and for cobipartite graphs, i.e., we can also give a matching lower bound example for these graph classes. Hence, our enumeration algorithms for interval graphs, forests, split graphs and cobipartite graphs are all optimal. The importance of our results stems from the fact that, for other types of domination problems, optimal enumeration algorithms are not always found.

Interestingly, we use a different form of analysis for the running times of our different algorithms, and the branchings had to be tailored and tweaked to obtain the intended optimality results. Our Roman dominating functions enumeration algorithm for trees and forests is distinctively different from the one for minimal dominating sets by Rote (SODA 2019). Our approach also allows to give concrete formulas for counting minimal Roman dominating functions on more concrete graph families like paths.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Problems, reductions and completeness; Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms

**Keywords and phrases** special graph classes, counting problems, enumeration problems, domination problems, Roman domination

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.6

**Related Version** *Full Version*: <https://arxiv.org/abs/2208.05261>



© Faisal N. Abu-Khzam, Henning Fernau, and Kevin Mann;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 6; pp. 6:1–6:15

Leibniz International Proceedings in Informatics



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

## 1 Introduction

ROMAN DOMINATION comes with a nice (hi)story, on how to position armies on the various regions to secure the Roman Empire with the smallest cost, measured in the number of armies. “To secure” means that either (1) a region  $r$  has at least one army or (2) a region  $r'$  neighboring  $r$  contains two armies, so that it can afford sending one army to the region  $r$  without diminishing  $r'$ 's self-defense capabilities.

It is easy to view ROMAN DOMINATION as a graph-theoretic problem, where the map is modeled as a graph. ROMAN DOMINATION has received notable attention in the last two decades [7, 17, 23, 26, 40, 41, 44, 48, 49, 51]. Relevant to our work is the development of exact algorithms: ROMAN DOMINATION can be solved in  $\mathcal{O}(1.5014^n)$  time (and space), see [40, 52, 54]. More combinatorial studies can be found in [16, 18, 25, 35, 39, 42, 43, 47, 55, 56, 57] as well as in the more recent chapter on Roman domination of [34]. Although independently introduced in [46], the *differential* of a graph is tightly related, see also [1, 8, 9, 10]. To briefly summarize all these findings, in many ways concerning complexity, ROMAN DOMINATION and DOMINATING SET behave exactly the same. There are two notable and related exceptions, as delineated in [2], concerning *extension problems* and *output-sensitive enumeration*.

*Extension problems* often arise from search-tree algorithms for their optimization counterpart as follows. Assume that a search-tree node corresponds to a partial solution (or pre-solution)  $U$  and instead of proceeding with the search-tree algorithm (by exploring all the possible paths from this node onward) we ask whether we can extend  $U$  to a meaningful solution  $S$ . In the case of DOMINATING SET, this means that  $S$  is an inclusion-wise minimal dominating set that contains  $U$ . Unfortunately, this EXTENSION DOMINATING SET problem and many similar problems are NP-hard, see [6, 12, 14, 15, 37, 38, 45]. Even worse: when parameterized by the “pre-solution size,” EXTENSION DOMINATING SET is one of the few problems known to be complete for the parameterized complexity class  $W[3]$ , as shown in [11]. This blocks any progress on the HITTING SET TRANSVERSAL PROBLEM by using extension test algorithms, which is the question whether all minimal hitting sets of a hypergraph can be enumerated with polynomial delay (or even output-polynomial) only. This question is open for four decades by now and is equivalent to several enumeration problems in logic, database theory and also to enumerating minimal dominating sets in graphs, see [22, 24, 29, 36].

By way of contrast and quite surprisingly, with an appropriate definition of the notion of minimality, the extension variant of ROMAN DOMINATION is solvable in polynomial time [3]. This was the key observation to show that enumerating all minimal Roman dominating functions is possible with polynomial delay. This triggered further interest in looking into enumerating minimal Roman dominating functions on graph classes, as also done in the case of DOMINATING SET, see [5, 20, 21, 30, 32, 33]. The basis of the output-sensitive enumeration result of [2] was several combinatorial observations. Here, we find ways how to use the underlying combinatorial ideas for non-trivial enumeration algorithms for minimal Roman dominating functions in split graphs, cobipartite graphs, interval graphs, forests and chordal graphs and for counting these exactly for paths. All these graph classes will be explained in separate sections below. These exploits constitute the main results of this paper. More details can be found at the end of the next section. Due to lack of space, further technical details can be found in [4]. We summarize known bases of lower and upper bounds on the number of minimal (Roman) dominating sets (resp. functions) in the next table; new results are shown with boxes; for matching bounds, only one number is displayed; c.f. [2, 21, 27, 33]. Polynomial delay is achievable for the mentioned special graph classes for enumerating minimal dominating sets [36, 38], but it is unclear how to combine these

approaches with good input-sensitive enumeration, while all input-sensitive results concerning minimal Roman dominating functions can also be implemented with polynomial delay, by interleaving extension tests with branching.

graph class:	general	chordal	split	interval	forests	cobipartite
domination	1.5704 / 1.7159	$\sqrt[3]{3} / 1.5048$	$\sqrt[3]{3}$	$\sqrt[3]{3}$	$\sqrt[13]{95}$	1.3195 / 1.3674
Roman dom.	1.7441 / 1.9332	$\sqrt{3} / 1.8940$	$\sqrt[3]{3}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt[3]{3}$

## 2 Definitions and Known Results

Let  $\mathbb{N} = \{1, 2, 3, \dots\}$  be the set of positive integers. For  $n \in \mathbb{N}$ , let  $[n] = \{m \in \mathbb{N} \mid m \leq n\}$ . We only consider undirected simple graphs. Let  $G = (V, E)$  be a graph. For  $U \subseteq V$ ,  $G[U]$  denotes the graph induced by  $U$ . For  $v \in V$ ,  $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$  denotes the *open neighborhood* of  $v$ , while  $N_G[v] := N_G(v) \cup \{v\}$  is the *closed neighborhood* of  $v$ .  $|N_G(v)|$  is called the *degree* of  $v$ ; a vertex of degree 1 is known as a *leaf*. We extend such set-valued functions  $X : V \rightarrow 2^V$  to  $X : 2^V \rightarrow 2^V$  by setting  $X(U) = \bigcup_{u \in U} X(u)$ . Subset  $D \subseteq V$  is a *dominating set*, or ds for short, if  $N_G[D] = V$ . For  $D \subseteq V$  and  $v \in D$ , define the *private neighborhood* of  $v \in V$  with respect to  $D$  as  $P_{G,D}(v) := N_G[v] \setminus N_G[D \setminus \{v\}]$ . A function  $f : V \rightarrow \{0, 1, 2\}$  is called a *Roman dominating function*, or rdf for short, if for each  $v \in V$  with  $f(v) = 0$ , there exists a  $u \in N_G(v)$  with  $f(u) = 2$ . Simplifying notation, we set  $V_i(f) := \{v \in V \mid f(v) = i\}$  for  $i \in \{0, 1, 2\}$ . The *weight*  $w_f$  of a function  $f : V \rightarrow \{0, 1, 2\}$  equals  $|V_1| + 2|V_2|$ . The ROMAN DOMINATION problem asks, given  $G$  and an integer  $k$ , if there exists an rdf of weight at most  $k$ . Connecting to the original motivation,  $G$  models a map of regions, and if the region vertex  $v$  belongs to  $V_i$ , then we place  $i$  armies on  $v$ .

For defining the problem EXTENSION ROMAN DOMINATION, we first need to define the order  $\leq$  on  $\{0, 1, 2\}^V$ : for  $f, g \in \{0, 1, 2\}^V$ , let  $f \leq g$  if and only if  $f(v) \leq g(v)$  for all  $v \in V$ . Thus, we extend the usual linear ordering  $\leq$  on  $\{0, 1, 2\}$  to functions mapping to  $\{0, 1, 2\}$  in a pointwise manner. We call a function  $f \in \{0, 1, 2\}^V$  a *minimal Roman dominating function* if and only if  $f$  is an rdf and there exists no rdf  $g$ ,  $g \neq f$ , with  $g \leq f$ . The weights of minimal rdFs can vary considerably. Consider for example a star  $K_{1,n}$  with center  $c$ . Then,  $f_1(c) = 2$ ,  $f_1(v) = 0$  otherwise;  $f_2(v) = 1$  for all vertices  $v$ ;  $f_3(c) = 0$ ,  $f_3(u) = 2$  for one  $u \neq c$ ,  $f_3(v) = 1$  otherwise, define three minimal rdFs with weights  $w_{f_1} = 2$ , and  $w_{f_2} = w_{f_3} = n + 1$ .

In [2], several combinatorial properties of minimal Roman dominating functions were derived that were central for obtaining a general algorithmic enumeration result and that are also important when studying special graph classes. This is summarized as follows.

► **Theorem 2.1.** *Let  $G = (V, E)$  be a graph,  $f : V \rightarrow \{0, 1, 2\}$  and abbreviate  $G' := G[V_0(f) \cup V_2(f)]$ . Then,  $f$  is a minimal rdf if and only if the following conditions hold:*

1.  $N_G[V_2(f)] \cap V_1(f) = \emptyset$ ,
2.  $\forall v \in V_2(f) : P_{G', V_2(f)}(v) \not\subseteq \{v\}$ , also called *privacy condition*, and
3.  $V_2(f)$  is a *minimal dominating set* of  $G'$ .

This combinatorial result has been the key to show a polynomial-time decision procedure for the extension problem (Given a graph  $G = (V, E)$ , a function  $f : V \rightarrow \{0, 1, 2\}$ , the question is if there is minimal rdf  $g$  with  $f \leq g$ ). It can also be used to design enumeration algorithms that are input-sensitive. The simplest exploit is to branch on all vertices whether or not a vertex should belong to  $V_2(f)$ . Once  $V_2(f)$  is fixed, its neighborhood will form  $V_0(f)$  and the remaining vertices will be  $V_1(f)$ . For better running times, this approach has to be refined, see Section 5. We obtain estimates of running times for branching algorithms as explained in [28], including an introduction into the Measure-and-Conquer analysis.

### 3 Enumerating Minimal RDFs in Split and in Cobipartite Graphs

A split graph  $G = (V, E)$  consists of a bipartition of  $V$  as  $C$  and  $I$ , such that  $C$  forms a clique and  $I$  is an independent set. Let  $f : V \rightarrow \{0, 1, 2\}$  be a minimal rdf of  $G$ . If  $V_2(f)$  contains both a vertex  $v_c$  from  $C$  and a vertex  $v_i$  from  $I$ , then  $v_i$  cannot find a private neighbor in  $G$ , contradicting the minimality of  $f$ . We can hence first branch to decide if  $V_2(f) \subseteq C$  or if  $V_2(f) \subseteq I$ . After dealing with the simple case that  $|V_2(f) \cap C| = 1$  separately, we can assume that all private neighbors of  $V_2(f) \subseteq C$  are in  $I$  and that all private neighbors of  $V_2(f) \subseteq I$  are in  $C$ . We will describe a simple branching algorithm in which we can assume to immediately delete vertices that are assigned the value 0, as they will be always dominated.

**Case 1.** One element of  $C$  is assigned a value of 2. We can guess this element in  $\mathcal{O}(n)$  and proceed as follows.

1. Elements of  $C$  with no neighbors in  $I$  are assigned a value of zero.
2. Pick  $v \in C$  with at least two neighbors in  $I$  and branch by either setting  $f(v) = 2$  and assign 0 to vertices in  $N(v) \cap I$  or  $f(v) = 0$  (this leads to the branching vector  $(3, 1)$ ).
3. When all elements of  $C$  have exactly one neighbor in  $I$ , pick some  $v \in C$  with  $N(v) \cap I = \{w\}$ . Distinguish two cases.
  - 3.1  $w$  has at least one other neighbor  $x \in C$ . Then either  $f(v) = 2, f(w) = f(x) = 0$  (in fact, all neighbors of  $w$  are assigned 0), or  $f(v) = 0$  (this leads to a  $(3, 1)$  branch).
  - 3.2  $N(w) = \{v\}$ : either  $f(v) = 2, f(w) = 0$  or  $f(v) = 0, f(w) = 1$  (this leads to the branching vector  $(2, 2)$ ).

**Case 2.** No element of  $C$  is assigned a value of 2.

1. Then any isolated element of  $I$  is automatically assigned a value of 1 and can be deleted. Moreover, any element of  $C$  with no neighbors in  $I$  is assigned a value of 1 and deleted.
2. Pick a vertex  $v$  of degree at least two in  $I$  and branch by either setting  $f(v) = 2$  and assigning 0 to all its neighbors or set  $f(v) = 1$  (this leads to the branching vector  $(3, 1)$ ).
3. When all elements of  $I$  are leaves, pick  $v \in I$  with  $N(v) \cap C = \{w\}$ . Distinguish 2 cases.
  - 3.1  $w$  has at least one more neighbor  $x \in I$ : either  $f(v) = 2, f(w) = 0, f(x) = 1$  or  $f(v) = 1$  (delete  $v$ ) (this leads to the branching vector  $(3, 1)$ ).
  - 3.2  $N(w) \cap I = \{v\}$ : either  $f(v) = 2, f(w) = 0$  or  $f(v) = f(w) = 1$  (this leads to the branching vector  $(2, 2)$ ).

Notice that the analysis of the recursion is very simple: an rdf  $f$  is gradually defined, and the branching vectors describe the number of newly defined vertices. The worst-case branching vector is  $(1, 3)$ , which leads to the following claim.

► **Proposition 3.1.** *All minimal rdfs in a split graph of order  $n$  are enumerable in  $\mathcal{O}^*(1.4656^n)$ .*

► **Remark 3.2.** For cobipartite graphs, a similar reasoning applies. Now, it could be possible that one vertex  $x$  of the bipartition side  $X$  finds its private neighbor  $p_x$  in  $X$  itself and that one vertex  $y$  of the other bipartition side  $Y$  finds its private neighbor  $p_y$  in  $Y$ , such that the edges  $xp_y$  and  $yp_x$  do not exist. If  $G$  contains no universal vertices, then irrespectively whether the  $V_2(f)$ -vertices lie only in  $X$  or in  $Y$ , there must be at least one other vertex in  $V_2(f)$  on the same side. But this means that they must find their private neighbors on the other side. The branching is hence analogous to the split graph case.

$$C_{P,2,n} = \begin{cases} n-1 & \text{if } n \in [2] \\ C_{P,\bar{2},n-2} & \text{if } n > 2 \end{cases} \quad C_{P,\bar{2},n} = \begin{cases} n & \text{if } n \in [3] \\ C_{P,\bar{2},n-1} + C_{P,2,n-2} + C_{P,2,n-3} + C_{P,\bar{2},n-3} & \text{if } n > 3 \end{cases}$$

■ **Figure 1** The mutual recurrences for determining the number of minimal rdfs on a path  $P_n$ .

The previous arguments are invalid in the case of bipartite graphs. Here, we conjecture that the general case is not really easier than the bipartite case, as with minimal ds enumeration.

However, we can boost our algorithm and its simple analysis to actually prove an optimal enumeration result. In order to do this, a rather straightforward refinement of the previous case analysis suffices. Together with Remark 3.2 as well as the trick of interleaving the branching with extension tests as described in [2], this refined branching algorithm proves:

► **Theorem 3.3.** *All minimal rdfs in a split graph or a cobipartite graph of order  $n$  can be enumerated in time  $\mathcal{O}^*(\sqrt[3]{3}^n)$ , using polynomial space and polynomial delay only.*

We can complement Theorem 3.3 by showing lower bound examples in the following that prove that our simple branching algorithm analysis is optimal for split and cobipartite graphs.

► **Theorem 3.4.** *There exist split and cobipartite graphs of order  $n$  with  $\Omega(\sqrt[3]{3}^n)$  many minimal rdfs.*

**Proof.** We consider the graph  $G_t = (C_t \cup I_t, E_t)$  with  $C_t = \{c_1, \dots, c_{2t}\}$ ,  $I_t = \{v_1, \dots, v_t\}$ ,  $3t = n = |C_t \cup I_t|$  and  $E_t = \binom{C_t}{2} \cup \{\{c_{2i-1}, v_i\}, \{c_{2i}, v_i\} \mid i \in [t]\}$  (for the cobipartite case,  $I_t$  is also a clique). Thus,  $v_i \in I_t$  has degree 2. If  $V_2(f) \subseteq C_t$ , there are three ways to Roman-dominate any  $v_i \in I_t$ ,  $c_{2i-1}, c_{2i} \in C_t$  with a minimal rdf  $f$ :  $f(c_{2i}) = 2$ ,  $f(c_{2i-1}) = f(v_i) = 0$  or  $f(c_{2i-1}) = 2$ ,  $f(c_{2i}) = f(v_i) = 0$  or  $f(v_i) = 1$ ,  $f(c_{2i-1}) = f(c_{2i}) = 0$  (resp.  $f(c_{2i-1}) = f(c_{2i}) = 1$ , if  $V_2(f) = \emptyset$ ). This yields  $3^t = \sqrt[3]{3}^n$  many minimal rdfs. There can be at most  $2^t = \sqrt[3]{2}^n$  minimal rdfs  $f$  on  $G_t$  with  $V_2(f) \subseteq I_t$ . Hence,  $G_t$  is a graph of order  $n = 3t$  that has  $\sqrt[3]{3}^n + \sqrt[3]{2}^n - 1 \in \Omega(\sqrt[3]{3}^n)$  many minimal rdfs. ◀

Minimal dominating sets in cobipartite graphs where all dominating set vertices belong to one clique only correspond to minimal rdfs with no vertex assigned 1. So, we can use our rdf enumeration algorithm to enumerate minimal dominating sets on cobipartite graphs. This improves on the hitherto best published algorithm from [19] but would be worse than [53].

## 4 Counting Minimal Roman Dominating Functions on Paths

The following is the main result of this section, devoted to counting.

► **Proposition 4.1.** *The number of minimal Roman dominating functions of a path  $P_n$  grows as  $\mathcal{O}^*(c_{RD,P}^n)$ , with  $c_{RD,P} \leq 1.6852$ .*

This should be compared with the recursion of Bród [13] that yields the following asymptotic behavior for the number of minimal dominating sets of a path with  $n$  vertices:

► **Corollary 4.2** (follows from [13]). *The number of minimal dominating sets of a path  $P_n$  grows as  $\mathcal{O}^*(c_{D,P}^n)$ , with  $c_{D,P} \leq 1.4013$ .*

As every minimal dominating set  $D \subseteq V$  of a graph  $G = (V, E)$  corresponds to the minimal rdf  $f : V \rightarrow \{0, 1, 2\}$  with  $V_2(f) = D$  and  $V_0(f) = V \setminus D$ , it is clear that  $c_{D,P} \leq c_{RD,P}$ .

**Proof of Proposition 4.1.** Let  $C_{P,n}$  count the number of minimal rdfs of a  $P_n$ . Furthermore, let  $C_{P,2,n}$  and  $C_{P,\bar{2},n}$  denote the number of minimal rdfs of a  $P_n$  where the first vertex is assigned 2, or where it is decided that the first vertex is not assigned 2, respectively. Clearly,  $C_{P,n} = C_{P,2,n} + C_{P,\bar{2},n}$ . Consider  $P_n = (V_n, E_n)$  with  $V_n = \{v_i \mid i \in [n]\}$  and  $E_n = \{v_i v_{i+1} \mid i \in [n-1]\}$ . Let  $n \geq 3$  and  $f : V_n \rightarrow \{0, 1, 2\}$  be a minimal rdf.

If  $f(v_1) = 2$ , then  $f(v_2) = 0$ . Also  $f(v_3) \neq 2$ , as  $v_1$  would not have a private neighbor but itself for  $f(v_3) = 2$ . This shows (including trivial initial cases) the left-hand side of Figure 1. If  $f(v_1) \neq 2$ , then we have two subcases: (a) if  $f(v_1) = 1$ , then we know  $f(v_2) \neq 2$ ; (b) if  $f(v_1) = 0$ , then  $f(v_2) = 2$  is enforced. But we know more compared to the initial situation:  $v_2$  has already a private neighbor, namely  $v_1$ . Thus, we have further possibilities for  $v_3$ :  $f(v_3) = 2$  or  $f(v_3) = 0$ . The first subcase is as before:  $v_3$  has no private neighbor. If  $f(v_3) = 0$ , then either  $f(v_4) = 2$  and  $v_4$  has no private neighbor, or  $f(v_4) \neq 2$ ; hence the recursions on the right-hand side of Figure 1. Keeping in mind that  $C_{P,n-3} = C_{P,2,n-3} + C_{P,\bar{2},n-3}$ , we see  $C_{P,n} = C_{P,2,n} + C_{P,\bar{2},n} = C_{P,\bar{2},n-2} + C_{P,\bar{2},n-1} + C_{P,2,n-2} + C_{P,n-3} = C_{P,\bar{2},n-1} + C_{P,n-2} + C_{P,n-3}$ . Conversely,  $C_{P,n} = C_{P,2,n} + C_{P,\bar{2},n} = C_{P,\bar{2},n-2} + C_{P,\bar{2},n}$ . Hence,

$$C_{P,n} = C_{P,\bar{2},n} + C_{P,\bar{2},n-2} = C_{P,\bar{2},n-1} + (C_{P,\bar{2},n-2} + C_{P,\bar{2},n-4}) + (C_{P,\bar{2},n-3} + C_{P,\bar{2},n-5}),$$

which gives, ignoring the cases for small values of  $n$ , the following single recursion:

$$C_{P,\bar{2},n} = C_{P,\bar{2},n-1} + C_{P,\bar{2},n-3} + C_{P,\bar{2},n-4} + C_{P,\bar{2},n-5} \approx 1.6852^n$$

As  $C_{P,n} = C_{P,\bar{2},n-2} + C_{P,\bar{2},n}$ , the same asymptotic behavior holds for  $C_{P,n}$ . ◀

We will further extend this result towards forests and towards interval graphs in the next sections, starting with a more general description of such branching algorithms.

## 5 A General Approach to Branching for Minimal RDFs

In this section, we sketch the general strategy that we apply for enumerating minimal rdfs. In most cases, the branching will look for a yet undecided vertex  $v$  (that we will call *active* henceforth) and will decide to label it with 2 in one branch and not to label it with 2 in the other branch. Now, in the first branch, we can say something about the neighbors of  $v$  as well: according to Theorem 2.1, they cannot be finally labelled with 1. We express this and similar properties by (always) splitting the vertex set  $V$  of the current graph  $G = (V, E)$  into:

- $A$ : active vertices. In the very beginning of the branching, all vertices are active.
- $\bar{V}_i$ : vertices that cannot be assigned a value of  $i$ ,  $i \in \{1, 2\}$ , due to previous decisions.
- $V_0$ : set of vertices assigned a value of zero that are not yet dominated.

Sometimes, the branching also considers a vertex from  $\bar{V}_1$ , which will be assigned 0 (and hence is deleted) in the branch when it is not assigned 2. We can also call extendibility tests before doing the branching in order to achieve polynomial delay; see [2].

Possibly, we can also (temporarily) have (and speak of) vertex sets  $V_i$  (with  $i \in \{1, 2\}$ ) with the meaning that each vertex in  $V_i$  is assigned the value  $i$ . Our algorithms will preserve the invariant that a vertex  $v \in \bar{V}_1$  must have a neighbor put into  $V_2$  (in the original graph), i.e.,  $N(v) \cap V_2 \neq \emptyset$ , which is a property that can be exploited in our analysis. Namely, a vertex is put into  $\bar{V}_1$  only if one of its neighbors has been put into  $V_2$ . However, notice that once the effect (mostly implied by Theorem 2.1) of putting a vertex  $v$  into  $V_i$  on its neighborhood  $N(v)$  has been taken care of, such a vertex  $v$  can be deleted from the “current graph” to simplify the considerations. More precisely, for  $i \in \{1, 2\}$ , our algorithms automatically delete vertices assigned a value of  $i$  after making sure the neighbors are placed in  $\bar{V}_{3-i}$ . It could happen

that the neighbor of a vertex  $w \in \overline{V_2}$  is assigned the value 2. Then,  $w$  must be assigned 0; as it is dominated, it can and will be deleted. Similarly, if the neighbor of a vertex  $w \in \overline{V_1}$  is assigned the value 1,  $w$  must be assigned 0 and is hence deleted. Only finally, it should be checked if a function  $f : V \rightarrow \{0, 1, 2\}$  that is constructed during branching is indeed a minimal rdf, as possibly some vertices assigned 2 do not have a private neighbor. During the course of our algorithm, whenever we speak of the degree of a vertex (in the current graph) in the following, we only count in neighbors in  $A \cup \overline{V_1} \cup \overline{V_2}$ . In most stages of our algorithms, we can assume  $V_1 = \emptyset$ , as we will explain.

Reduction rules are an important ingredient of any branching algorithm, as also shown in [28]. We will make use of the following reduction rules. Similar rules appeared in [2].

- ▶ Reduction Rule 5.1. If  $v \in \overline{V_2}$  with  $N(v) \subseteq \overline{V_2}$ , then set  $f(v) = 1$  and delete  $v$ .
- ▶ Reduction Rule 5.2. If  $v \in \overline{V_1}$  with  $N(v) \subseteq \overline{V_1}$ , then set  $f(v) = 0$  and delete  $v$ .
- ▶ Reduction Rule 5.3. If  $v \in A$  with  $N(v) \subseteq \overline{V_1}$ , then put  $v$  into  $\overline{V_2}$ .

▶ **Lemma 5.1.** *The three presented reduction rules are sound.*

In contrast to our approach in Section 3, we will now perform a Measure-and-Conquer analysis of the branching algorithms that we will describe. As a measure, we take

$$\mu(A, \overline{V_1}, \overline{V_2}, V_0) = |A| + \omega_1 |\overline{V_1}| + \omega_2 |\overline{V_2}|$$

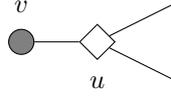
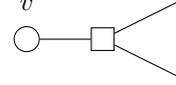
for the “current graph” with vertex set partitioned as  $A \cup \overline{V_1} \cup \overline{V_2} \cup V_0$ . Hence, whenever we measure our graph, we can assume  $V_1 = \emptyset$ . In the beginning of the algorithm,  $A = V$  and  $\overline{V_1} = \overline{V_2} = V_0 = V_1 = \emptyset$ . To explain the work of the reduction rules, consider an isolated vertex (in the very beginning). The reduction rules will first move it into  $\overline{V_2}$  and then into  $\overline{V_1}$  to finally delete it. We will choose the constants  $\omega_1, \omega_2 \in [0, 1]$  to assess the running times of our algorithms best possible, hence also delivering upper bounds on the number of minimal rdfs of graphs of order  $n$  belonging to a specific graph class.

Concerning the reduction rules, we can easily observe that their application will never increase the measure. We will list in the following several branching rules (for the different graph classes) and we always assume that the rules are carried out in the given order.

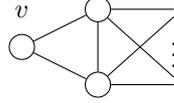
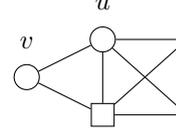
## 6 Enumerating Minimal RdFs on Interval Graphs and Forests

Recall that an *interval graph* can be described as the intersection graph of a collection of intervals on the real line. This means that the vertices correspond to intervals and that there is an edge between two such vertices if the intervals have a non-empty intersection. We assume in the following that  $G = (V, E)$  is an interval graph with the interval representation  $\mathcal{I} = \{I_v := [l_v, r_v]\}_{v \in V}$ , i.e.,  $l_v$  is the left border and  $r_v$  is the right border of the interval representing the vertex  $v$ . We call  $v \in U$  *leftmost in*  $U \subseteq V$  if it is a vertex from  $U$  that has the smallest value of  $r_u$  among all vertices in  $U$ . A vertex leftmost in  $V$  is simply called *leftmost*. Notice that this notion of a leftmost vertex will be used in many places in the rules exhibited in the following and is not available in the setting of general graphs as investigated in [3] but relies on the interval graph structure. Our algorithm always branches on the leftmost vertex. Then, it simply considers all cases. We now present more details.

The reduction rules from Section 5 imply that each vertex in  $v \in A$  has at least one neighbor in  $A \cup \overline{V_2}$ . Concerning the measure, we will have  $\omega_1 = 1$  and set  $\omega_2 = \omega = 0.57$ . We will present the branching rules that constitute the backbone of our algorithm for enumerating minimal rdfs on interval graphs. We often provide illustrations of the different branching scenarios. In our figures, we adhere to the following drawing conventions:

(a)  $v$  is already dominated.(b)  $v \in A$  has neighbors in  $\overline{V_2}$ .

■ **Figure 2** Branching Rules 6.1 and 6.2. Here (and elsewhere in these illustrations) we only sketch important parts of a subgraph, not necessarily covering all cases of the rules within the drawings.

(a)  $v \in A$  has neighbors in  $A$ .(b)  $v \in A$  has neighbors in  $\overline{V_2}$  and in  $A$ .

■ **Figure 3** Branching Rules 6.3 and 6.4.

- ○ are vertices in  $A$ , ● are vertices in  $\overline{V_1}$ , ⊙ are vertices in  $A \cup \overline{V_1}$ .
- □ are vertices in  $\overline{V_2}$ .
- ◇ are vertices in  $A \cup \overline{V_2}$ , for which the exact set is not further defined.
- ◆ are vertices in  $A \cup \overline{V_1} \cup \overline{V_2}$ , for which the exact set is not further defined.

► **Branching Rule 6.1.** Let  $v$  be the leftmost vertex in  $\overline{V_1}$  and let  $u$  be the leftmost vertex in  $N(v) \cap (A \cup \overline{V_2})$  and branch as follows: (1) Put  $v$  in  $V_0$ . (2) Put  $v$  in  $V_2$  and  $u$  in  $V_0$ .

► **Lemma 6.1.** *The branching of Branching Rule 6.1 is a complete case distinction. Moreover, it leads at worst to the following branching vector:  $(1, 1 + \omega)$ .*

One can formulate and prove similar lemmas for the other branching rules that we present; see [4]. The branching vectors and branching numbers are summarized in Table 1.

► **Branching Rule 6.2.** Let  $v$  be the leftmost vertex in  $(A \cup \overline{V_2})$ . If  $v \in A$  and  $N(v) \cap A = \emptyset$  hold, branch as follows: (1) Put  $v$  in  $V_2$  and  $N(v) \cap \overline{V_2}$  in  $V_0$ . (2) Put  $v$  in  $V_1$ .

► **Branching Rule 6.3.** Let  $v$  be leftmost in  $(A \cup \overline{V_2})$ . If  $v \in A$  and  $|N(v) \cap A| \geq 2$  hold, branch as follows: (1) Put  $v$  in  $V_2$  and all vertices in  $N(v) \cap A$  into  $V_0$ . (2) Put  $v$  in  $\overline{V_2}$ .

► **Branching Rule 6.4.** Let  $v$  be the leftmost vertex in  $(A \cup \overline{V_2})$ . If  $v \in A$ ,  $|N(v) \cap \overline{V_2}| \geq 1$  and  $|N(v) \cap A| = 1$  with  $u \in N(v) \cap A$  hold, then branch: (1) Put  $v$  in  $V_2$ ,  $N(v) \cap (\{u\} \cup \overline{V_2})$  in  $V_0$ . (2) Put  $u$  in  $V_2$  and  $\{v\} \cup (N(v) \cap \overline{V_2})$  in  $V_0$ . (3) Put  $v$  in  $V_0$  and  $u$  in  $\overline{V_2}$ .

► **Branching Rule 6.5.** Let  $v$  be the leftmost vertex in  $(A \cup \overline{V_2})$ . If  $N[v] \cap A = \{v, u\}$  with  $N[v] \cap \overline{V_2} = \emptyset$  and  $|N(u) \cap A| \geq 3$ , branch as follows:  
(1) Put  $v$  in  $V_2$ ,  $u$  in  $V_0$  and  $N(u) \setminus \{v\}$  in  $\overline{V_2}$ . (2) Put  $v$  in  $\overline{V_2}$ .

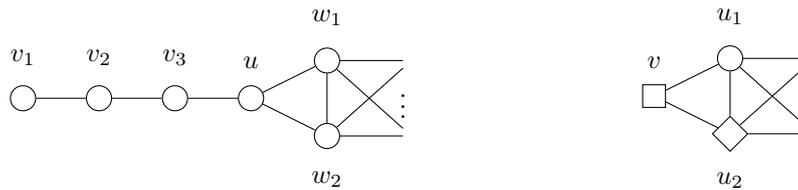
► **Branching Rule 6.6.** Let  $v_1$  be the leftmost vertex in  $(A \cup \overline{V_2})$ . If  $N[v_1] \cap A = \{v_1, v_2\}$  with  $N[v_1] \cap \overline{V_2} = \emptyset$ ,  $N(v_2) \cap A = \{v_1, v_3\}$  and if there exists a  $u \in N(v_3)$  such that  $N(u) = \{v_3\}$ , then branch as follows: (1) Put  $v_1$  in  $V_2$ ,  $v_2$  in  $V_0$  and  $v_3$  in  $\overline{V_2}$ . (2) Put  $v_1$  in  $V_1$ ,  $v_2$  in  $\overline{V_2}$ . (3) Put  $v_2$  in  $V_2$  and  $v_1, v_3$  in  $V_0$  and  $u$  in  $V_1$ . (4) Put  $v_2, v_3$  in  $V_2$  and  $v_1, u$  in  $V_0$ .

► **Branching Rule 6.7.** Let  $v_1$  be the leftmost vertex in  $(A \cup \overline{V_2})$ , such that  $N[v_1] \cap A = \{v_1, v_2\}$ , with  $N[v_2] \cap \overline{V_2} = \emptyset$  and  $N(v_2) \cap (A \cup \overline{V_2}) = \{v_1, v_3\}$ . If there is a  $u$  leftmost in  $A \setminus \{v_1, v_2, v_3\}$ , with  $\{v_3\} \subsetneq N(u)$ , then branch as follows: (1) Put  $v_1$  in  $V_2$  and  $v_2$  in  $V_0$  and  $v_3$  in  $\overline{V_2}$ . (2) Put  $v_1$  in  $V_1$ ,  $v_2$  in  $\overline{V_2}$ . (3) Put  $v_1$  in  $V_0$ ,  $v_2$  in  $V_2$  and  $v_3$  in  $\overline{V_2}$ . (4) Put  $v_2, v_3$  in  $V_2$  and  $v_1, u$  in  $V_0$  and  $N(u) \setminus \{v_3\}$  in  $\overline{V_2}$ .



(a)  $v_1 \in A$  has only one neighbor in  $A$  which has degree bigger than 2. (b)  $v_1, v_2, v_3 \in A$  is a path and  $v_3$  has a leaf neighbor.

■ **Figure 4** Branching Rules 6.5 and 6.6.



(a)  $v_1, v_2, v_3 \in A$  is a path and there exists a  $u \in A \cap N(v_3)$  with one more neighbor. (b)  $v \in \overline{V_2}$  is the leftmost vertex.

■ **Figure 5** Rules 6.7 and 6.8.

► **Branching Rule 6.8.** Let  $v$  be the leftmost vertex in  $(A \cup \overline{V_2})$ . If  $v \in \overline{V_2}$ , branch like: (1) For each  $u \in N(v) \cap A$ :  $u$  in  $V_2$  and  $N[v] \setminus \{u\}$  into  $V_0$ . (2) Put  $v$  in  $V_1$  and  $N(v) \cap A$  in  $\overline{V_2}$ .

► **Theorem 6.2.** All minimal rdfs of an interval graph of order  $n$  can be enumerated in time  $\mathcal{O}^*(\sqrt{3}^n)$ , with polynomial delay and in polynomial space.

This result is optimal, as there are interval graphs that have  $\sqrt{3}^n$  many minimal rdfs, namely collections of paths on two vertices:  $x - y$  can be Roman-dominated by  $f(x) = f(y) = 1$  or by assigning two to one vertex and zero to the other one, i.e., we get three possibilities per two vertices. For optimally enumerating minimal ds in interval graphs, see [31].

Recall that a *forest* is an acyclic undirected graph. A branching scenario that is similar to, but slightly more complex than, that of interval graphs can be used for forests (see [4]).

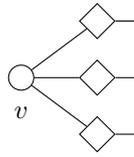
► **Theorem 6.3.** A forest of order  $n$  has at most  $\sqrt{3}^n$  many minimal rdfs. They can also be enumerated in time  $\mathcal{O}^*(\sqrt{3}^n)$ , with polynomial delay and in polynomial space.

■ **Table 1** Branching scenarios on interval graphs.

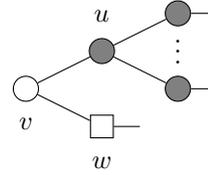
rule	branching vector	branching number
6.1 & 6.2	$(1, 1 + \omega)$	1.7314
6.3	$(3, 1 - \omega)$	1.6992
6.4	$(2 + \omega, 2 + \omega, 2 - \omega)$	1.6829
6.5	$(4 - 2\omega, 1 - \omega)$	1.7274
6.6	$(3 - \omega, 2 - \omega, 4, 4)$	1.6877
6.7	$(3 - \omega, 2 - \omega, 3, 5 - \omega)$	1.7315
6.8	$(\underbrace{\omega +  N(v) \cap A , \dots, \omega +  N(v) \cap A }_{ N(v) \cap A  \text{ many times}}, \omega + (1 - \omega) \cdot  N(v) \cap A )$	$\leq \sqrt{3} \leq 1.7321$

■ **Table 2** Branching rules and their vectors and numbers for chordal graphs; **worst cases in red.**

Rule	branching vector	branching number
7.1	$(1 - \omega_2, 1 + 3 \min(1 - \omega_1, \omega_2))$	<b>1.8940</b>
7.2	$(1 + \omega_1 + \omega_2, 1 - \omega_2)$	1.8014
7.3	$(\omega_1, \omega_1 + 2\omega_2)$	<b>1.8940</b>
7.4	$(\omega_1, 2 - \omega_1 + \min(1 - \omega_1, \omega_2))$	<b>1.8940</b>
7.5 & 7.16	$(\omega_1, 2\omega_1 + \omega_2)$	1.7915
7.6	$(\omega_1 + \omega_2, \omega_1 + \omega_2)$	1.8321
7.7 & 7.14	$(1 + \omega_2, 1 + \omega_2)$	never worse than Branching Rule 7.6
7.8	$(1 + \omega_2 + \min(1 - \omega_2, \omega_1), 1)$	1.6181
7.9	$(2, 1 - \omega_2)$	1.8471
7.10	$(1 + \omega_2, 1)$	1.779
7.11	$(1 + \omega_1 + 2(1 - \omega_2), \omega_1)$	1.5743
7.12	$(1 + 2\omega_2, 1)$	never worse than Branching Rule 7.10
7.13	$(2 + \omega_2, 1 - \omega_2)$	1.7249
7.15 & 7.17	$(2 - \omega_1 + \omega_2, 2 + \omega_2, 2 - \omega_2)$	1.8005



(a)  $v \in A$  has at least 3 neighbors in  $A \cup \overline{V_2}$ .



(b)  $v \in A$  has one neighbor  $w \in \overline{V_2}$  and at least one neighbor in  $\overline{V_1}$  that has only further neighbors in  $\overline{V_1}$ .

■ **Figure 6** Branching Rules 7.1 and 7.2.

This result is again optimal, as there are forests that have  $\sqrt{3}^n$  many minimal rdfs, namely collections of  $P_2$ . A similar optimality result was obtained by Rote [50] for enumerating minimal dominating sets in forests by using different techniques: there are (at most)  $\sqrt[13]{95}^n$  many of them in forests of order  $n$ .

## 7 Enumerating Minimal RDFs in Chordal Graphs

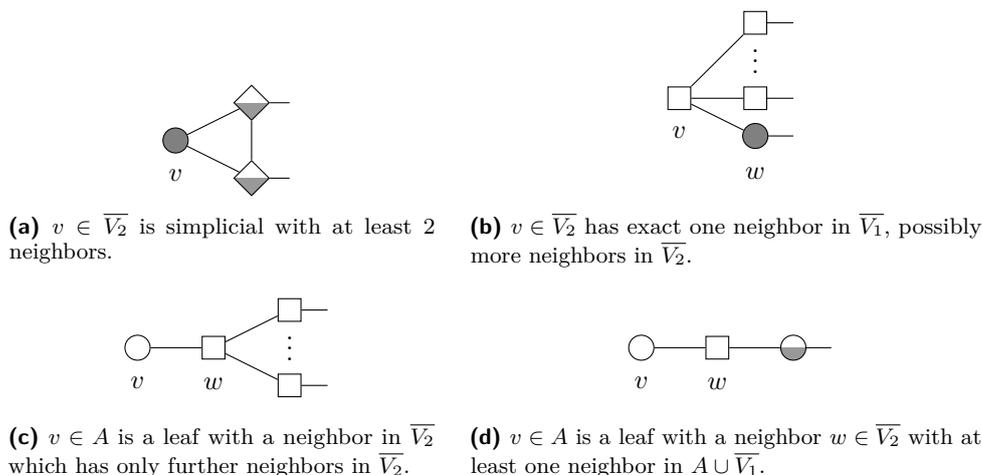
Recall that a graph is *chordal* if the only induced cycles it might contain have length three. In this quite technical section, we explain the following result whose optimality is open.

► **Theorem 7.1.** *All minimal Roman dominating functions of a chordal graph of order  $n$  can be enumerated with polynomial delay and in polynomial space in time  $\mathcal{O}(1.8940^n)$ .*

We are following the general approach sketched in Section 5. We adopt as a measure  $\mu = |A| + \omega_1 |\overline{V_1}| + \omega_2 |\overline{V_2}|$ . To obtain our result, we set  $\omega_1 = 0.710134$  and  $\omega_2 = 0.434799$ .

Initially, all vertices are in  $A$ . Each branching rule assumes the preceding rules have been applied exhaustively and none of their conditions is applicable anymore. We omit stating correctness lemmas and lemmas concerning branching vectors but refer to Table 2. These lemmas are in general quite simple.

► **Branching Rule 7.1.** If  $v \in A$  has at least three neighbors in  $A \cup \overline{V_2}$ , then we branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Add  $v$  to  $\overline{V_2}$ .



■ **Figure 7** Branching Rules 7.5, 7.6, 7.7 and 7.8.

From now on, we can assume that a vertex from  $A$  of degree at least 3 has a neighbor in  $\overline{V}_1$ .

► **Branching Rule 7.2.** If  $v \in A$  has at least one neighbor  $w$  in  $\overline{V}_2$  and at least one neighbor  $u$  in  $\overline{V}_1$  such that all neighbors of  $u$  (but  $v$  and possibly  $w$ ) are in  $\overline{V}_1$ , then we branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Add  $v$  to  $\overline{V}_2$ .

Knowing (by our invariants) that elements of  $\overline{V}_1$  are guaranteed to have neighbors in  $V_2$ , the next two branching rules apply to some elements of  $\overline{V}_1$  (illustration can be found in [4]):

► **Branching Rule 7.3.** If  $v \in \overline{V}_1$  has at least two neighbors in  $\overline{V}_2$ , then we branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Set  $f(v) = 0$  and delete  $v$ .

► **Branching Rule 7.4.** If  $v \in \overline{V}_1$  has at least three neighbors in  $A \cup \overline{V}_2$  then we branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Set  $f(v) = 0$  and delete  $v$ .

From now on, we discuss branching on simplicial vertices (or sometimes on vertices in the neighborhood of simplicial vertices as in [5]).

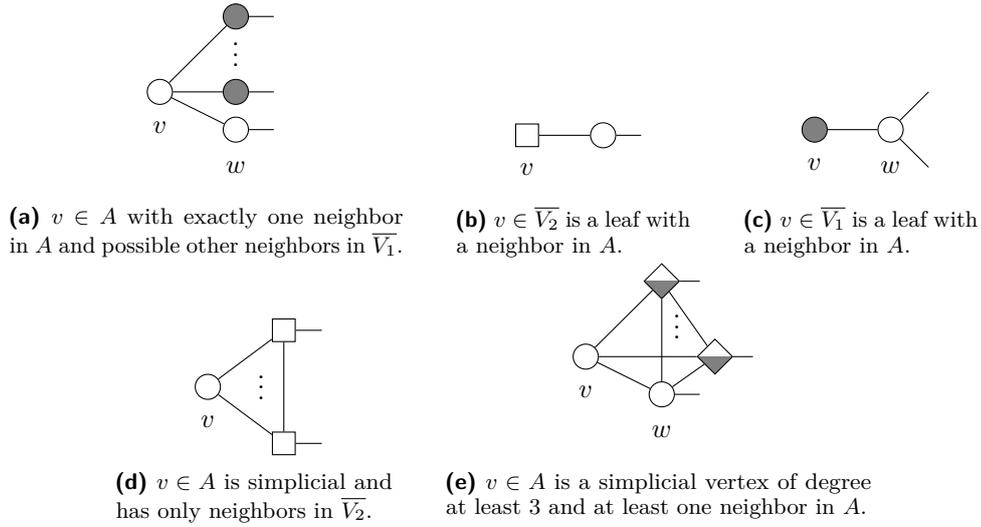
► **Observation 7.2.** *Simplicial vertices in  $\overline{V}_1$  can only have neighbors in  $A \cup \overline{V}_2 \cup \overline{V}_1$ . As we already considered vertices in  $\overline{V}_1$  with  $\geq 3$  neighbors in  $A \cup \overline{V}_2$ , in the following branchings, a vertex in  $\overline{V}_1$  has  $\leq 2$  neighbors in  $A \cup \overline{V}_2$ , not both of them in  $\overline{V}_2$  due to Branching Rule 7.3.*

► **Branching Rule 7.5.** If  $v \in \overline{V}_1$  is simplicial and of degree at least two, then branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Set  $f(v) = 0$  and delete  $v$ .

► **Observation 7.3.** *We note that an isolated pair of adjacent leaves, say  $v, w$ , give rise to a path, which has already been studied. However, assuming previous branching rules have resulted in such a path, the worst case is when  $v \in \overline{V}_2$  and  $w \in \overline{V}_1$ . To see this, note that if both  $v$  and  $w$  are in  $\overline{V}_2$  or both in  $\overline{V}_1$ , they would be deleted by Reduction Rules 5.1 or 5.2.*

► **Branching Rule 7.6.** If  $v \in \overline{V}_2$  is a vertex with exactly one neighbor  $w \in \overline{V}_1$  and possibly more neighbors in  $\overline{V}_2$ , then we branch as follows: (1) Set  $f(w) = 2$ ,  $f(v) = 0$  and update the neighbors of  $w$  accordingly. (2) Set  $f(w) = 0$  and  $f(v) = 1$  and delete  $v, w$ .

► **Branching Rule 7.7.** Let  $v \in A$  with  $N(v) = \{w\}$ ,  $w \in \overline{V}_2$ , with  $N(w) \setminus \{v\} \subseteq \overline{V}_2$ . Then, branch as follows: (1) Set  $f(v) = 2$  and  $f(w) = 0$ . (2) Set  $f(v) = f(w) = 1$  and delete  $v, w$ .



■ **Figure 8** Branching Rules 7.9, 7.10, 7.11, 7.12 and 7.13.

► **Branching Rule 7.8.** If  $v \in A$  with  $N(v) = \{w\}$  and  $w \in \overline{V_2}$  and if there is at least one further neighbor of  $w$  that belongs to  $A \cup \overline{V_1}$ , then we branch as follows: (1) Set  $f(v) = 2$  and  $f(w) = 0$  and update all neighbors of  $w$  to  $\overline{V_2}$  or to  $V_0$ . (2) Set  $f(v) = 1$  and delete  $v$ .

The following rule again deals with a leaf vertex as a special case.

► **Branching Rule 7.9.** Let  $v \in A$  with  $N(v) \cap A = \{w\}$  and  $N(v) \setminus \{w\} \subseteq \overline{V_1}$ . Then, branch: (1) Set  $f(v) = 2$  and  $f(w) = 0$ , update all neighbors of  $w$  to  $\overline{V_2}$  or to  $V_0$ . (2) Add  $v$  to  $\overline{V_2}$ .

► **Branching Rule 7.10.** If  $v \in \overline{V_2}$  with  $N(v) = \{w\}$ ,  $w \in A$ , then we branch as follows: (1) Set  $f(w) = 2$  and  $f(v) = 0$ ; update  $N(w)$  accordingly. (2) Add  $w$  to  $\overline{V_2}$  and set  $f(v) = 1$ .

► **Branching Rule 7.11.** If  $v \in \overline{V_1}$  with  $N(v) = \{w\}$ ,  $w \in A$  and  $|N(w) \cap A| = 2$ , then branch: (1) Set  $f(v) = 2$ ,  $f(w) = 0$  and put the neighbors of  $w$  into  $\overline{V_2}$ . (2) Set  $f(v) = 0$  and delete  $v$ .

► **Branching Rule 7.12.** If  $v \in A$  is simplicial, of degree  $\geq 2$  with  $N(v) \subset \overline{V_2}$ , then branch: (1) Set  $f(v) = 2$  and assign zero to all its neighbors (delete  $N[v]$ ). (2) Set  $f(v) = 1$  and delete  $v$ .

► **Branching Rule 7.13.** If  $v \in A$  is simplicial, with  $|N(v)| \geq 2$  and  $N(v) \cap A \neq \emptyset$ , then we branch as follows: (1) Set  $f(v) = 2$  and update the neighbors accordingly. (2) Add  $v$  to  $\overline{V_2}$ .

Finally, we consider simplicial vertices in  $\overline{V_2}$  of degree  $\geq 2$ , now covering the remaining cases.

► **Branching Rule 7.14.** Let  $v \in \overline{V_2}$  be a simplicial vertex of degree two with a neighbor  $w \in A$ . If the other neighbor  $w'$  of  $v$  is in  $\overline{V_2}$ , then we branch as follows: (1) Set  $f(w) = 2$  and  $f(v) = f(w') = 0$ . (2) Add  $w$  to  $\overline{V_2}$ , set  $f(v) = 1$  and delete  $v$ .

► **Branching Rule 7.15.** If  $v \in \overline{V_2}$  is simplicial with two neighbors  $w, w' \in A$ , then we branch as follows: (1) Set  $f(w) = 2$ ,  $f(v) = 0$  and add  $w'$  to  $\overline{V_1}$ . (2) Set  $f(w') = 2$  and  $f(w) = f(v) = 0$ . (3) Add  $w$  and  $w'$  to  $\overline{V_2}$  and set  $f(v) = 1$ .

► **Branching Rule 7.16.** If  $v \in \overline{V_2}$  is simplicial, of degree at least two, with a neighbor  $w$  such that  $N[w] \setminus \{v\} \subseteq \overline{V_1}$ , then we branch as follows: (1) Set  $f(w) = 2$ ,  $f(v) = 0$  and delete  $N[v]$ . (2) Set  $f(w) = 0$  and delete it.

► **Branching Rule 7.17.** If  $v \in \overline{V_2}$  is simplicial, with neighbors  $w, w' \in \overline{V_1}$  s.t.  $N[w] \subset N[w']$ , then branch: (1) Set  $f(w') = 2$ ,  $f(v) = f(w) = 0$ . (2) Set  $f(w') = 0$  and delete it.

► **Lemma 7.4.** *Our rules cover all possible cases for chordal graphs.*

It remains open whether enumeration on chordal graphs can be improved further, so we hereby pose it as an open problem, or whether one can obtain a higher lower bound, which might also be a gap-improvement on general graphs. So far, the best lower bound for general graphs is a collection of  $C_5$ 's [2], which is clearly not a chordal graph. The worst-case example for chordal graphs is a collection of  $P_2$ 's, see Section 4 and our discussions on interval graphs.

---

## References

- 1 F. N. Abu-Khzam, C. Bazgan, M. Chopin, and H. Fernau. Data reductions and combinatorial bounds for improved approximation algorithms. *Journal of Computer and System Sciences*, 82(3):503–520, 2016.
- 2 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. Technical Report 2204.04765, Cornell University, ArXiv/CoRR, 2022. doi:10.48550/arXiv.2204.04765.
- 3 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. In M. A. Bekos and M. Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG*, volume 13453 of *LNCS*, pages 1–15. Springer, 2022. doi:10.1007/978-3-031-15914-5\_1.
- 4 F. N. Abu-Khzam, H. Fernau, and K. Mann. Roman census: Enumerating and counting Roman dominating functions on graph classes. Technical Report 2208.05261, Cornell University, ArXiv/CoRR, 2022. arXiv:2208.05261.
- 5 F. N. Abu-Khzam and P. Heggernes. Enumerating minimal dominating sets in chordal graphs. *Information Processing Letters*, 116(12):739–743, 2016.
- 6 C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K.-M. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.
- 7 S. Benecke. Higher order domination of graphs. Master's thesis, Department of Applied Mathematics of the University of Stellenbosch, South Africa, <http://dip.sun.ac.za/~vuuren/Theses/Benecke.pdf>, 2004.
- 8 S. Bermudo and H. Fernau. Computing the differential of a graph: hardness, approximability and exact algorithms. *Discrete Applied Mathematics*, 165:69–82, 2014.
- 9 S. Bermudo and H. Fernau. Combinatorics for smaller kernels: The differential of a graph. *Theoretical Computer Science*, 562:330–345, 2015.
- 10 S. Bermudo, H. Fernau, and J. M. Sigarreta. The differential and the Roman domination number of a graph. *Applicable Analysis and Discrete Mathematics*, 8:155–171, 2014.
- 11 T. Bläsius, T. Friedrich, J. Lischied, K. Meeks, and M. Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022.
- 12 M. Bonamy, O. Defrain, M. Heinrich, and J.-F. Raymond. Enumerating minimal dominating sets in triangle-free graphs. In R. Niedermeier and C. Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *LIPIcs*, pages 16:1–16:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 13 D. Bród. On the number of minimal dominating sets in some classes of trees. *Int. J. Contemp. Math. Sciences*, 6:503–506, 2011.
- 14 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. Extension of some edge graph problems: Standard and parameterized complexity. In L. A. Gasieniec, J. Jansson, and C. Levcopoulos, editors, *Fundamentals of Computation Theory - 22nd International Symposium, FCT*, volume 11651 of *LNCS*, pages 185–200. Springer, 2019.
- 15 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. Abundant extensions. In T. Calamoneri and F. Corò, editors, *Algorithms and Complexity - 12th International Conference, CIAC*, volume 12701 of *LNCS*, pages 3–17. Springer, 2021. doi:10.1007/978-3-030-75242-2\_1.

- 16 E. W. Chambers, B. Kinnersley, N. Prince, and D. B. West. Extremal problems for Roman domination. *SIAM Journal of Discrete Mathematics*, 23:1575–1586, 2009.
- 17 M. Chapelle, M. Cochefert, J.-F. Couturier, D. Kratsch, M. Liedloff, and A. Perez. Exact algorithms for weak Roman domination. In T. Lecroq and L. Mouchard, editors, *Combinatorial Algorithms - 24th International Workshop, IWOCA*, volume 8288 of *LNCS*, pages 81–93. Springer, 2013.
- 18 M. Chellali, T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and A. A. McRae. A Roman domination chain. *Graphs and Combinatorics*, 32(1):79–92, 2016.
- 19 J.-F. Couturier, P. Heggernes, P. van 't Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *LNCS*, pages 202–213. Springer, 2012.
- 20 J.-F. Couturier, P. Heggernes, P. van 't Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. *Theoretical Computer Science*, 487:82–94, 2013.
- 21 J.-F. Couturier, R. Letourneur, and M. Liedloff. On the number of minimal dominating sets on some graph classes. *Theoretical Computer Science*, 562:634–642, 2015.
- 22 N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019.
- 23 P. A. Dreyer. *Applications and Variations of Domination in Graphs*. PhD thesis, Rutgers University, New Jersey, USA, 2000.
- 24 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
- 25 O. Favaron, H. Karami, R. Khoeilar, and S. M. Sheikholeslami. On the Roman domination number of a graph. *Discrete Mathematics*, 309(10):3447–3451, 2009.
- 26 H. Fernau. ROMAN DOMINATION: a parameterized perspective. *International Journal of Computer Mathematics*, 85:25–38, 2008.
- 27 F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.
- 28 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.
- 29 A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM Journal of Discrete Mathematics*, 31(1):63–100, 2017.
- 30 P. A. Golovach, P. Heggernes, M. M. Kanté, D. Kratsch, and Y. Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 199:30–36, 2016.
- 31 P. A. Golovach, P. Heggernes, M. Moustapha Kanté, D. Kratsch, and Y. Villanger. Minimal dominating sets in interval graphs and trees. *Discrete Applied Mathematics*, 216:162–170, 2017.
- 32 P. A. Golovach, P. Heggernes, and D. Kratsch. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science*, 630:63–75, 2016.
- 33 P. A. Golovach, D. Kratsch, M. Liedloff, and M. Y. Sayadi. Enumeration and maximum number of minimal dominating sets for chordal graphs. *Theoretical Computer Science*, 783:41–52, 2019.
- 34 T. W. Haynes, S.T. Hedetniemi, and M. A. Henning, editors. *Topics in Domination in Graphs*, volume 64 of *Developments in Mathematics*. Springer, 2020.
- 35 S. T. Hedetniemi, R. R. Rubalcaba, P. J. Slater, and M. Walsh. Few compare to the great Roman empire. *Congressus Numerantium*, 217:129–136, 2013.
- 36 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal of Discrete Mathematics*, 28(4):1916–1929, 2014.

- 37 M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In F. Dehne, J.-R. Sack, and U. Stege, editors, *Workshop on Algorithms and Data Structures, WADS*, volume 9214 of *LNCS*, pages 446–457. Springer, 2015.
- 38 M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In E. W. Mayr, editor, *International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2015*, volume 9224 of *LNCS*, pages 138–153. Springer, 2016.
- 39 T. Kraner Šumenjak, P. Pavlič, and A. Tepeh. On the Roman domination in the lexicographic product of graphs. *Discrete Applied Mathematics*, 160(13-14):2030–2036, 2012.
- 40 M. Liedloff. *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles: domination, variantes et généralisations*. PhD thesis, Université Paul Verlaine - Metz, France, 2007.
- 41 M. Liedloff, T. Kloks, J. Liu, and S.-L. Peng. Efficient algorithms for Roman domination on some classes of graphs. *Discrete Applied Mathematics*, 156(18):3400–3415, 2008.
- 42 C.-H. Liu and G. J. Chang. Roman domination on 2-connected graphs. *SIAM Journal of Discrete Mathematics*, 26(1):193–205, 2012.
- 43 C.-H. Liu and G. J. Chang. Upper bounds on Roman domination numbers of graphs. *Discrete Mathematics*, 312(7):1386–1391, 2012.
- 44 C.-H. Liu and G. J. Chang. Roman domination on strongly chordal graphs. *Journal of Combinatorial Optimization*, 26(3):608–619, 2013.
- 45 A. Mary. *Énumération des dominants minimaux d’un graphe*. PhD thesis, LIMOS, Université Blaise Pascal, Clermont-Ferrand, France, November 2013.
- 46 J. L. Mashburn, T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and P. J. Slater. Differentials in graphs. *Utilitas Mathematica*, 69:43–54, 2006.
- 47 B. P. Mobaraky and S. M. Sheikholeslami. Bounds on Roman domination numbers of graphs. *Matematichki Vesnik*, 60:247–253, 2008.
- 48 A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, Roman domination and other dominating set variants. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Networking and Mobile Computing, IFIP 17<sup>th</sup> World Computer Congress — TC1 Stream / 2<sup>nd</sup> IFIP International Conference on Theoretical Computer Science IFIP TCS*, pages 280–291. Kluwer, 2002. Also available as Technical Report 365, ETH Zürich, Institute of Theoretical Computer Science, 10/2001.
- 49 S.-L. Peng and Y.-H. Tsai. Roman domination on graphs of bounded treewidth. In *The 24th Workshop on Combinatorial Mathematics and Computation Theory*, pages 128–131, 2007.
- 50 G. Rote. The maximum number of minimal dominating sets in a tree. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1201–1214. SIAM, 2019.
- 51 W. Shang, X. Wang, and X. Hu. Roman domination and its variants in unit disk graphs. *Discrete Mathematics, Algorithms and Applications*, 2(1):99–106, 2010.
- 52 Z. Shi and K. M. Koh. Counting the number of minimum Roman dominating functions of a graph. Technical report, ArXiv / CoRR, abs/1403.1019, 2014. [arXiv:1403.1019](https://arxiv.org/abs/1403.1019).
- 53 I. B. Skjørten. Faster enumeration of minimal connected dominating sets in split graphs. Master’s thesis, Department of Informatics, University of Bergen, Norway, June 2017.
- 54 J. M. M. van Rooij. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. PhD thesis, Universiteit Utrecht, The Netherlands, 2011.
- 55 H.-M. Xing, X. Chen, and X.-G. Chen. A note on Roman domination in graphs. *Discrete Mathematics*, 306(24):3338–3340, 2006.
- 56 F. Xueliang, Y. Yuansheng, and J. Baoqi. Roman domination in regular graphs. *Discrete Mathematics*, 309(6):1528–1537, 2009.
- 57 I. G. Yero and J. A. Rodríguez-Velázquez. Roman domination in Cartesian product graphs and strong product graphs. *Applicable Analysis and Discrete Mathematics*, 7:262–274, 2013.



# Counting Computations with Formulae: Logical Characterisations of Counting Complexity Classes

Antonis Achilleos   

Department of Computer Science, Reykjavik University, Iceland

Aggeliki Chalki   

Department of Computer Science, Reykjavik University, Iceland

---

## Abstract

---

We present quantitative logics with two-step semantics based on the framework of quantitative logics introduced by Arenas et al. (2020) and the two-step semantics defined in the context of weighted logics by Gastin & Monmege (2018). We show that some of the fragments of our logics augmented with a least fixed point operator capture interesting classes of counting problems. Specifically, we answer an open question in the area of descriptive complexity of counting problems by providing logical characterisations of two subclasses of  $\#P$ , namely  $\text{SpanL}$  and  $\text{TotP}$ , that play a significant role in the study of approximable counting problems. Moreover, we define logics that capture  $\text{FPSPACE}$  and  $\text{SpanPSPACE}$ , which are counting versions of  $\text{PSPACE}$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Complexity theory and logic; Theory of computation  $\rightarrow$  Complexity classes

**Keywords and phrases** descriptive complexity, quantitative logics, counting problems,  $\#P$

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.7

**Related Version** *Full Version:* <https://doi.org/10.48550/arXiv.2304.10334>

**Funding** This work has been funded by the projects “Open Problems in the Equational Logic of Processes (OPEL)” (grant no. 196050), “Mode(l)s of Verification and Monitorability” (MoVeMnt) (grant no 217987) of the Icelandic Research Fund, and the Basic Research Program PEVE 2020 of the National Technical University of Athens.

**Acknowledgements** The authors would like to thank Stathis Zachos and Aris Pagourtzis for fruitful discussions and Luca Aceto for sound advice. We also thank the anonymous reviewers for their suggestions and constructive comments.

## 1 Introduction

We examine counting problems from the viewpoint of descriptive complexity. We present a quantitative logic with a least fixed point operator and two-step semantics. In the first step, given a structure, a formula generates a set. In the second step, a quantitative interpretation results from the cardinality of that set. These semantics allow us to use a uniform approach to identify logical fragments that capture several counting complexity classes.

In 1979, Valiant introduced the complexity class  $\#P$  in his seminal paper [32] and used it to characterise the complexity of computing the permanent function.  $\#P$  is the class of functions that count accepting paths of non-deterministic poly-time Turing machines, or, equivalently, the number of solutions to problems in NP. For example,  $\#\text{SAT}$  is the function that, on input a formula  $\varphi$  in CNF, returns the number of satisfying assignments of  $\varphi$ . Since then, counting complexity has played an important role in computational complexity theory.

Descriptive complexity provides characterisations of complexity classes in terms of the logic needed to express their problems. The Büchi–Elgot–Trakhtenbrot theorem [9, 15, 31] characterising regular languages in terms of Monadic Second-Order logic and Fagin’s theorem [17], which states that Existential Second-Order logic captures NP, are two fundamental results in this area. Another prominent result was the introduction of the class  $\text{MaxSNP}$  [29],



© Antonis Achilleos and Aggeliki Chalki;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 7; pp. 7:1–7:15

Leibniz International Proceedings in Informatics



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

which has played a central role in the study of the hardness of approximation for optimization problems [6]. Moreover, descriptive complexity is an interesting and active research field with more recent results in the logical characterisation of the class  $P$  [20], dynamic complexity [33], symmetric linear programs [7], and counting complexity [5, 12], among others.

As in the case of optimization problems, an interesting, long-standing question is the logical characterisation of approximable counting problems. This is also a meaningful line of research since very few counting problems can be computed exactly in polynomial time. In the case of counting problems, the appropriate notion of approximability is the existence of a fully polynomial randomized approximation scheme (fpras). We denote the class of approximable counting problems by  $FPRAS$  [13, 8].

A counting class is considered *robust* if it has either natural complete problems or nice closure properties. Two robust subclasses of  $\#P$  defined in terms of Turing Machines (TMs), are of great significance in the quest for a characterisation of approximable counting problems. The first one is  $TotP$ , which contains all self-reducible counting problems whose decision version is in  $P$ . It is noteworthy that  $TotP$  is not contained in  $FPRAS$ , unless  $RP = NP$  [8], but almost all known approximable counting problems belong to  $TotP$  (see e.g. [23, 22, 27]). The second class, namely  $SpanL$  [2], is contained in  $TotP$ , and it consists of the functions that count different outputs of non-deterministic log-space *transducers*, i.e. TMs with output. To the best of our knowledge,  $SpanL$  is the only counting class so far defined in terms of TMs, that, despite containing  $\#P$ -complete problems [2], contains only approximable problems [4].

**Our contribution.** Our main objective is to provide logical characterisations of the classes  $SpanL$  and  $TotP$ , which was posed as an open question in [5]. To this end, we introduce a variant of the quantitative logics from [5]. Our two-step semantic definition is the key difference between our approach and that in [5]. The first step is an *intermediate semantics*, where the meaning of a formula is given as a set of strings that, intuitively, represent computation paths. In the second step, a concrete semantics associates with each formula the size of the set resulting from the intermediate semantics. Gastin et al. follow an analogous approach for weighted logics in [18], to give a connection to weighted automata.

In Section 4, we introduce logics equipped with least fixed point formulae that capture “span-classes” of restricted space, namely  $SpanL$  and  $SpanPSPACE$ , in a natural way (Theorems 4.7 and 4.13). When we consider such classes, we are interested in counting the number of different outputs produced by a transducer. Semantics that map the set of quantitative formulae to  $\mathbb{N}$  interpret every accepting path as a contributing unit. Then, by evaluating the sum of formulae as the sum of natural numbers, one can sum up the accepting paths of a TM. On the other hand, when a formula is evaluated as a set of output strings and the sum of formulae as the union of sets, they can count the number of different TM outputs.

We also consider two classes, namely  $\#PSPACE$  and  $TotP$ , which contain functions that count the accepting or all paths of TMs with restricted resources, respectively. Using our alternative semantics, a computation path can be encoded as a sequence of configurations visited by the TM along that path – in other words, its computation history – so that different paths are mapped to different sequences. In Section 5, we provide a logical characterisation of the class of functions that count the number of accepting paths of poly-space TMs, namely  $\#PSPACE$  [25] (Theorem 5.3), which coincides with  $FSPACE$ , i.e. the class of poly-space computable functions.  $FSPACE$  has already been characterised by a logic with a partial fixed point [5]. Interestingly, the logic we define here uses a least fixed point. In Section 6, we introduce a quantitative logic that captures  $TotP$  (Theorem 6.6). In Section 7, we discuss how to obtain two least fixed point logics that capture  $NL$  and  $PSPACE$  by specialising the semantics. We believe that the semantics we propose in this paper can contribute insight to the study of counting complexity classes.

**Related work.** Arenas et al. and Saluja et al. give logical characterisations of  $\#P$  in [30, 5]. The authors of [30] substitute existential quantification over second-order variables of  $\exists SO$  with counting second-order variables. The work in [5] incorporated counting into the syntax of the logic by introducing Quantitative Second-Order logic (QSO), a logic for quantitative functions, which is based on the framework of weighted logics [11, 18, 1]. There has been progress in characterising counting classes with respect to their approximability in the context of descriptive complexity. Saluja et al. defined the classes  $\#\Sigma_1$  and  $\#\Sigma_2$  in [30], and proved that they contain only problems that admit an fpras. A more recent variant of  $\#\Sigma_1$  [12] is also a subclass of FPRAS. The class  $\#\mathsf{R}\Pi_1$  [13] is conjectured to contain problems which are neither as hard to approximate as  $\#\mathsf{SAT}$  nor admit an fpras, and it was used to classify Boolean  $\#\mathsf{CSP}$  with respect to their approximability [14]. Since NP-complete problems cannot have approximable counting versions unless  $\mathsf{RP} = \mathsf{NP}$  [13], Arenas et al. suggested in [5] to examine robust classes of counting problems with an easy decision version. The papers [5, 8] defined such counting classes and examined them with respect to the approximability of their problems. There is also work on logics that capture superclasses of  $\#P$ , namely  $\mathsf{SpanP}$  [24] and  $\mathsf{FPSPACE}$  [25]. Compton and Grädel were the first to characterise  $\mathsf{SpanP}$  in [10], followed by Arenas et al. in [5], where they also introduced a logic that captures  $\mathsf{FPSPACE}$ . Finally, in [12], Durand et al. introduced a framework for the descriptive complexity of arithmetic circuit classes.

## 2 Preliminaries

**Turing machines.** A (*two-tape non-deterministic*) Turing machine (TM)  $N$  is a quintuple  $N = (\mathcal{Q}, \Sigma, \delta, q_0, q_F)$ , where  $\mathcal{Q}$  is a set of states,  $\Sigma = \{0, 1\}$  is the alphabet,  $\delta \subseteq (\mathcal{Q} \times (\Sigma \cup \{-\})^2) \times (\mathcal{Q} \times (\Sigma \cup \{-\})) \times \{L, R\}^2$  is the transition relation,  $q_0$  is the initial state, and  $q_F$  is the final accepting state. We assume the TM  $N$  has a read-only input tape and a work tape that it can read and write on.  $L$  and  $R$  in a transition designate that the respective tape head moves to the left or right. A *configuration*  $c$  of  $N$  encodes a snapshot of the computation of  $N$  and is defined in the usual way (see e.g. [28]). We can apply a compatible transition to a configuration to result in a new configuration in the expected way. W.l.o.g. we assume that every TM has a binary computation tree: any configuration is compatible with zero, one or two transitions. In the latter case, we call these transitions, the *left* and *right* non-deterministic transition. A *transducer*  $M$  is a TM with a write-only output tape, on which a string over  $\Sigma$  is written from left to right. The output of a computation is *valid* if  $M$  stops in the accepting state. A TM or transducer is called *deterministic* if at every configuration at most one transition can be applied. By restricting the time or space resources of a TM or transducer in the usual way, we can obtain an NPTM (non-deterministic poly-time TM), an NL-transducer (non-deterministic log-space transducer) etc.

We say that  $f$  is *computable in polynomial time* (resp. logarithmic/polynomial space), if there is a deterministic polynomial-time (resp. log-space/poly-space) transducer  $M$ , such that for every  $x \in \Sigma^*$ ,  $f(x)$  is the valid output of  $M$  on input  $x$ . We define the functions that count paths (resp. outputs) of a TM (resp. transducer) as follows.

► **Definition 2.1.** Let  $M$  be a Turing machine and  $T$  a transducer. We define functions  $acc_M, tot_M, span_T : \Sigma^* \rightarrow \mathbb{N} \cup \{+\infty\}$ , such that for every  $x \in \Sigma^*$ :

- (a)  $acc_M(x) = \#(\text{accepting computation paths of } M \text{ on input } x)$ ,
- (b)  $tot_M(x) = \#(\text{computation paths of } M \text{ on input } x) - 1$ ,
- (c)  $span_T(x) = \#(\text{different valid outputs of } T \text{ on input } x)$ .

**Classes of counting problems.** The classes defined in Definition 2.2 are already known, except for SpanPSPACE, which is presently defined.

- **Definition 2.2** ([2, 27, 25]). (a)  $\text{SpanL} = \{\text{span}_M : \Sigma^* \rightarrow \mathbb{N} \mid M \text{ is an NL-transducer}\}$ ,
- (b)  $\text{TotP} = \{\text{tot}_M : \Sigma^* \rightarrow \mathbb{N} \mid M \text{ is an NPTM}\}$ ,
- (c)  $\text{FPSPACE} = \{f : \Sigma^* \rightarrow \mathbb{N} \mid f \text{ is computable in polynomial space}\}$ ,
- (d)  $\#\text{PSPACE} = \{\text{acc}_M : \Sigma^* \rightarrow \mathbb{N} \mid M \text{ is a non-deterministic poly-space TM}\}$ .
- (e)  $\text{SpanPSPACE} = \{\text{span}_M : \Sigma^* \rightarrow \mathbb{N} \mid M \text{ is a non-deterministic poly-space transducer}\}$ .

► **Remark 2.3.** Note that in the definition of TotP, one is subtracted from the total number of paths so that a function can take the zero value. Since a TotP function  $f$  can be associated with an NPTM  $M$  that has a binary computation tree,  $f(x) = \text{tot}_M(x) = \#(\text{branchings of } M \text{ on input } x)$ , where a branching is an occurrence of a configuration on the computation tree, where  $M$  makes a non-deterministic choice.

► **Remark 2.4.** For the class SpanL, note that, by the pigeonhole principle, an NL-transducer has infinitely many accepting paths if and only if the length of its accepting runs is not bounded by a polynomial. It then makes sense to attach a clock that imposes a polynomial-time bound to each NLTM, as suggested in [2]. In this way, every NLTM is also an NPTM with a finite number of computation paths. Similarly, we assume that a clock that imposes an exponential-time bound can be attached to a non-deterministic poly-space TM.

► **Proposition 2.5** ([2, 27, 25]).  $\text{SpanL} \subseteq \text{TotP} \subseteq \#\text{P} \subsetneq \text{FPSPACE} = \#\text{PSPACE} \subseteq \text{SpanPSPACE}$ . The first two inclusions are proper unless  $\text{P} = \text{NP}$ .

The *decision version* of a function  $f : \Sigma^* \rightarrow \mathbb{N}$  is  $\{x \mid f(x) > 0\}$ . We say that a function  $f : \Sigma^* \rightarrow \mathbb{N}$  is *self-reducible* if its value on an instance can be recursively computed by evaluating  $f$  on a polynomial number of smaller instances. A formal definition of self-reducibility can be found in [3]. TotP can be characterised as the closure under parsimonious reductions of the class of self-reducible  $\#\text{P}$  functions whose decision version is in P [27].

► **Example 2.6.** Consider the problem of counting independent sets of all sizes in a graph  $G$ , denoted by  $\#\text{IS}$ . Let  $M$  be the NPTM that makes the following computation: given  $G_{i-1}$  and  $v_i, \dots, v_n$ ,  $M$  non-deterministically chooses to add vertex  $v_i$  to the independent set or not, and defines  $G_i$  to be either  $G_{i-1}$  where  $v_i$ , all its neighbours, and all edges adjacent to them have been removed, or  $G_{i-1}$  where  $v_i$  and its adjacent edges have been removed, respectively. Then,  $M$  recursively continues on  $G_i$  and  $v_{i+1}, \dots, v_n$ . Consider  $M'$  that on input  $G = \langle V = \{v_1, \dots, v_n\}, E \rangle$  simulates  $M$  on  $G$  and  $v_1, \dots, v_n$ , and has also an additional dummy path. Then,  $\#\text{IS}(G) = \#(\text{paths of } M' \text{ on input } G) - 1$ .

**Logics.** A relational vocabulary  $\sigma = \{\mathcal{R}_1^{k_1}, \dots, \mathcal{R}_m^{k_m}\}$  is a finite set of relation symbols. Each relation symbol  $\mathcal{R}_i$  has a positive integer  $k_i$  as its designated arity. A *finite structure*  $\mathcal{A} = \langle A, R_1, \dots, R_m \rangle$  over  $\sigma$  consists of a finite set  $A$ , which is called the *universe* of  $\mathcal{A}$  and relations  $R_1, \dots, R_m$  of arities  $k_1, \dots, k_m$  on  $A$ , which are interpretations of the corresponding relation symbols. We may write that  $\text{arity}(R_i) = k_i$  or that  $R_i$  is a  $k_i$ -ary relation. The *size of the structure*, denoted by  $|\mathcal{A}|$  or  $|A|$ , is the size of its universe. A *finite ordered structure* is a finite structure with an extra relation  $\leq$ , which is interpreted as a total order on the elements of the universe. In sequel,  $\mathcal{A}$  denotes a finite ordered structure unless otherwise specified. For convenience we use letters  $B, C, R, S$ , and so on, to denote both relation symbols and their interpretations. For example, the vocabulary of binary strings is  $\sigma_{bs} = \{\leq^2, B^1\}$ . Binary string  $x = 00101$  corresponds to the structure  $\mathcal{A} = \langle \{0, 1, \dots, 4\}, \leq, B = \{2, 4\} \rangle$ , where relation  $B$  represents the positions where  $x$  is one. Moreover,  $|\mathcal{A}| = 5$ .

First-order formulae over  $\sigma$  are defined in the usual way, using first-order variables that range over the universe of a structure, the relation symbols from  $\sigma$ , equality, the logical operators  $\wedge, \vee, \neg, \rightarrow$ , and first-order quantifiers  $\forall x$  and  $\exists x$ . For convenience and clarity, we omit function and constant symbols from the syntax of **FO**, but we include  $\top$ , which is the logical constant for truth. A first-order formula with no free variable occurrences is called a first-order *sentence*, where an occurrence of  $x$  is free if it does not lie in the scope of either  $\exists x$  or  $\forall x$ . In addition to the syntax of **FO**, **S0** includes and quantifies over second-order variables that range over relations, are denoted by uppercase letters, and each of them has an arity. **S0** includes formulae of the form  $X(x_1, \dots, x_k)$ , where  $X$  is a second-order variable of arity  $k$ , and  $x_1, \dots, x_k$  are first-order variables. The fragment of **S0** consisting only of existential second-order formulae is called existential second-order logic and is abbreviated as  $\exists\text{S0}$ . We use the usual  $\mathcal{A}, v, V \models \varphi$  interpretation of an **S0**-formula  $\varphi$ , given a structure  $\mathcal{A}$  and first- and second-order assignments  $v$  and  $V$ , respectively. If  $\varphi$  has no free first- or second-order variables,  $v$  or  $V$ , respectively, can be omitted. We refer the reader to [16] for a more extensive presentation of **FO** and **S0**.

The logical symbols of Quantitative Second-Order logic, denoted by **QSO**, include all the logical symbols of **S0** and the quantitative quantifiers  $\Sigma$  and  $\Pi$  for sum and product quantification, respectively. The arity of a second-order variable  $X$  is denoted by  $\text{arity}(X)$ . When we write logic  $\Lambda$  over  $\sigma$ , we mean the set of  $\Lambda$  formulae over  $\sigma$ . The set of **QSO** formulae over  $\sigma$  are defined by the following grammar:

$$\alpha ::= \varphi \mid s \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma x.\alpha \mid \Pi x.\alpha \mid \Sigma X.\alpha \mid \Pi X.\alpha \quad (1)$$

where  $\varphi$  is an **S0** formula over  $\sigma$ ,  $s \in \mathbb{N}$ ,  $x$  is a first-order and  $X$  a second-order variable. A formula  $\alpha$  in **QSO** is a sentence if every variable occurrence in  $\alpha$  is bound by a first-order, second-order, or quantitative quantifier. The evaluation of a **QSO** formula  $\alpha$  is a function  $\llbracket \alpha \rrbracket$  that on input  $\mathcal{A}, v$ , and  $V$  returns a number in  $\mathbb{N}$ . We refer the reader to [5, p. 5] for the definition of the semantics of **QSO** formulae. When  $\alpha$  is a sentence,  $\llbracket \alpha \rrbracket(\mathcal{A})$  is used to denote  $\llbracket \alpha \rrbracket(\mathcal{A}, v, V)$  for any  $v, V$ . We say that  $f \in \text{QSO}$  if there exists  $\alpha \in \text{QSO}$  such that  $f(\text{enc}(\mathcal{A})) = \llbracket \alpha \rrbracket(\mathcal{A})$ , for every  $\mathcal{A}$ . Note that **QSO** is a set of logical formulae, whereas  $\text{QSO}$  is a class of functions. For every logic  $\Lambda$ , we can define a corresponding class of functions as above, and we denote it by  $\Lambda$ .

► **Definition 2.7.** *A logic  $\Lambda$  captures a complexity class  $\mathsf{C}$ , and equivalently  $\mathsf{C} = \Lambda$ , over finite ordered structures over  $\sigma$ , if the following two conditions hold:*

1. *For every  $f \in \mathsf{C}$ , there is a sentence  $\alpha \in \Lambda$ , such that  $f(\text{enc}(\mathcal{A})) = \llbracket \alpha \rrbracket(\mathcal{A})$  for every finite ordered structure  $\mathcal{A}$  over  $\sigma$ .*
2. *For every sentence  $\alpha \in \Lambda$ , there is a function  $f \in \mathsf{C}$ , such that  $\llbracket \alpha \rrbracket(\mathcal{A}) = f(\text{enc}(\mathcal{A}))$  for every finite ordered structure  $\mathcal{A}$  over  $\sigma$ .*

*Moreover,  $\Lambda$  captures  $\mathsf{C}$  over finite ordered structures if  $\Lambda$  captures  $\mathsf{C}$  over finite ordered structures over  $\sigma$ , for every  $\sigma$ .*

For example,  $\Sigma\text{QSO}(\text{FO}) = \#\text{P}$  over finite ordered structures [5], where  $\Sigma\text{QSO}(\text{FO})$  is the set of **QSO** formulae that  $\Pi$  is not allowed and  $\varphi$  in (1) is restricted to be an **FO** formula.

Triples  $(\mathcal{A}, v, V)$  can be encoded in space polynomial in  $|\mathcal{A}|$  using a standard mapping from finite ordered structures to strings over  $\{0, 1\}$  (see for example [26, Chapter 6]). We assume that a TM  $M$  takes as input the encoding of  $\mathcal{A}$  (or  $(\mathcal{A}, v, V)$ ), denoted by  $\text{enc}(\mathcal{A})$  (resp.  $\text{enc}(\mathcal{A}, v, V)$ ), even if we write  $M(\mathcal{A})$  (resp.  $M(\mathcal{A}, v, V)$ ) for the sake of brevity.

In all cases that we consider in this paper, the initial configuration of a TM is **FO** definable [21] and therefore, to prove that  $\Lambda$  captures  $\mathsf{C}$ , it suffices to verify conditions 1 and 2 in Definition 2.7 for  $f(\text{enc}(\mathcal{A}, v, V)) = \llbracket \alpha \rrbracket(\mathcal{A}, v, V)$ , where  $v, V$  encode the initial

## 7:6 Counting Computations with Formulae

$$\begin{aligned}
\text{Expl}[x](\mathcal{A}, v, V) &= \{v(x)\} \\
\text{Expl}[X](\mathcal{A}, v, V) &= \{V(X)\} \\
\text{Expl}[\varphi](\mathcal{A}, v, V) &= \begin{cases} \{\varepsilon\}, & \text{if } \mathcal{A}, v, V \models \varphi \\ \emptyset, & \text{otherwise} \end{cases} \\
\text{Expl}[\alpha_1 + \alpha_2](\mathcal{A}, v, V) &= \text{Expl}[\alpha_1](\mathcal{A}, v, V) \cup \text{Expl}[\alpha_2](\mathcal{A}, v, V) \\
\text{Expl}[\alpha_1 \cdot \alpha_2](\mathcal{A}, v, V) &= \text{Expl}[\alpha_1](\mathcal{A}, v, V) \circ \text{Expl}[\alpha_2](\mathcal{A}, v, V) \\
\text{Expl}[\Sigma y.\alpha](\mathcal{A}, v, V) &= \bigcup_{a \in A} \text{Expl}[\alpha](\mathcal{A}, v[a/y], V) \\
\text{Expl}[\Sigma Y.\alpha](\mathcal{A}, v, V) &= \bigcup_{B \subseteq A^k} \text{Expl}[\alpha](\mathcal{A}, v, V[B/Y])
\end{aligned}$$

■ **Table 1** Intermediate semantics of  $\Sigma\text{SO}(\underline{\Lambda})$  formulae.

configuration of a TM that corresponds to  $f$ . Finally, we often use that (a)  $\mathcal{A}, v, V \models \varphi$  can be decided in deterministic logarithmic space, if  $\varphi$  is an FO formula, and in deterministic polynomial space, if  $\varphi \in \text{SO}$ , for every finite structure  $\mathcal{A}$  [21], and (b) given  $\mathcal{A}$ , the lexicographic order on  $k$ -tuples over  $A$  induced by  $\leq$  is FO expressible and is also denoted by  $\leq$ .

### 3 The quantitative logic $\Sigma\text{SO}(\underline{\Lambda})$

The logic  $\Sigma\text{SO}(\underline{\Lambda})$  over  $\sigma$ , where  $\Lambda \in \{\text{FO}, \text{SO}\}$ , is defined by the following grammar:

$$\alpha ::= x \mid X \mid \varphi \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \Sigma y.\alpha \mid \Sigma Y.\alpha \quad (2)$$

where  $\varphi$  is in  $\Lambda$ ,  $x, y$  are first-order variables, and  $X, Y$  are second-order variables. The syntax of logic  $\Sigma\text{SO}(\underline{\Lambda})$  is the same as that of  $\Sigma\text{QSO}(\underline{\Lambda})$ , where a formula can also be a first- and second-order variable, but not a number  $s \in \mathbb{N}$ .  $\Sigma\text{FO}(\underline{\Lambda})$  is the fragment of  $\Sigma\text{SO}(\underline{\Lambda})$  in which  $\Sigma$  is not allowed over second-order variables. We say that a  $\Sigma\text{SO}(\underline{\Lambda})$  formula is  $x$ -free (resp.  $X$ -free) if it is given by grammar (2) without  $x$  (resp.  $X$ ).

► **Notation Remark 3.1.** We denote  $X \cdot \varphi(X)$  (or  $\varphi(X) \cdot X$ ) by  $\varphi(\underline{X})$ .

We define the semantics of the logic  $\Sigma\text{SO}(\underline{\Lambda})$  in two phases: a formula  $\alpha$  is mapped to a set of strings. Then, the semantic interpretation of formula  $\alpha$  is defined to be the size of this set. Formally,  $\llbracket \alpha \rrbracket(\mathcal{A}, v, V) = |\text{Expl}[\alpha](\mathcal{A}, v, V)|$ , where  $\text{Expl}[\alpha](\mathcal{A}, v, V)$  is recursively defined in Table 1. Expl stands for Explicit and we call  $\text{Expl}[\alpha](\mathcal{A}, v, V)$  the *intermediate semantic interpretation* of formula  $\alpha$ . Note that  $\cup$  and  $\circ$  between sets of strings have replaced sum and multiplication of natural numbers, respectively, in the semantics of QSO.  $S_1 \cup S_2$  is the union of  $S_1$  and  $S_2$ , whereas  $S_1 \circ S_2$  is *concatenation* of sets of strings lifted from the concatenation operation on strings, that is  $S_1 \circ S_2 = \{x \circ y \mid x \in S_1, y \in S_2\}$ . For example,  $\{\varepsilon, a_1, a_2a_3\} \circ \{\varepsilon, a_2a_3\} = \{\varepsilon, a_2a_3, a_1, a_1a_2a_3, a_2a_3a_2a_3\}$ , where  $\varepsilon$  denotes the empty string. In specific, if one of  $S_1, S_2$  is  $\emptyset$ , then  $S_1 \circ S_2 = \emptyset$ .

► **Notation Remark 3.2.** For a finite set  $K$ ,  $K^* := \bigcup_{n \in \mathbb{N}} K^n$  denotes the set of strings over  $K$ ,  $\mathcal{P}(K^*)$  the powerset of  $K^*$ , and  $\varepsilon$  the empty string. For an  $\mathcal{A}$  over  $\sigma$ ,  $\mathcal{R}_k := \mathcal{P}(A^k)$  denotes the set of relations on  $A$  of arity  $k$ .

► **Remark 3.3.** Note that for a formula  $\alpha \in \Sigma\text{SO}(\underline{\Lambda})$  and  $s \in \text{Expl}[\alpha](\mathcal{A}, v, V)$ , we have that  $s \in (A \cup \bigcup_{i \in \mathbb{N}} \mathcal{R}_i)^*$ . In this paper, we consider logics that are either  $X$ -free or  $x$ -free, and so for a formula  $\alpha$  in some of these logics and  $s \in \text{Expl}[\alpha](\mathcal{A}, v, V)$ , either  $s \in A^*$  or  $s \in (\bigcup_{i \in \mathbb{N}} \mathcal{R}_i)^*$ , respectively.

The length of  $\alpha$ , denoted by  $|\alpha|$ , is defined as the length of  $\alpha$  as a string of symbols, boolean formulae and sum operators are treated as one symbol. The length of  $s \in A^* \cup (\bigcup_{i \in \mathbb{N}} \mathcal{R}_i)^*$ , denoted by  $|s|$ , is the standard length of strings. It is not hard to define an *encoding*  $\text{enc}(s)$  of  $s$ , such that  $|\text{enc}(s)| \leq |s| \cdot \log |A|$ , if  $s \in A^*$ , and  $|\text{enc}(s)| \leq |s| \cdot |A|^k$ , if  $s \in (\bigcup_{1 \leq i \leq k} \mathcal{R}_i)^*$ .

► **Lemma 3.4.** *Let  $\alpha$  be a  $\Sigma\text{SO}(\underline{\Lambda})$  formula over  $\sigma$ . For every finite ordered structure  $\mathcal{A}$  over  $\sigma$ ,  $v$ , and  $V$ , and every  $s \in \text{Expl}[\alpha](\mathcal{A}, v, V)$ ,  $|s| \leq |\alpha|$ . Moreover, (a) if  $\alpha$  is an  $X$ -free formula, then  $|\text{enc}(s)| \leq |\alpha| \cdot \log |A|$ , and (b) if  $\alpha$  is an  $x$ -free formula, then  $|\text{enc}(s)| \leq |\alpha| \cdot \text{poly}(|A|)$ .*

### 3.1 The logic $\Sigma\text{SO}(\underline{\Lambda})$ with recursion

By adding a function symbol  $f$  to the syntax of  $\Sigma\text{SO}(\underline{\Lambda})$ , we obtain formulae defined below:

$$\beta ::= x \mid X \mid \varphi \mid f(x_1, \dots, x_k) \mid (\beta + \beta) \mid (\beta \cdot \beta) \mid \Sigma y. \beta \mid \Sigma Y. \beta \quad (3)$$

where  $f$  is a *first-order function symbol* with  $\text{arity}(f) = k$ , and  $x_1, \dots, x_k$  are first-order variables, also denoted by  $\vec{x}$ . In like manner, we can add a *second-order function symbol* to  $\Sigma\text{SO}(\underline{\Lambda})$ . In particular, we consider only second-order function symbols of arity 1, i.e. of the form  $f(X)$ , where  $X$  is a second-order variable. A  $\Sigma\text{SO}(\underline{\Lambda})$  formula  $\beta(X, f)$  with a second-order function symbol  $f(Y)$  is called *arity-consistent* when it has at most one free second-order variable  $X$ , where  $X$  has the same arity as  $Y$ . We fix an arity  $k$  for the first-order function symbol, or the argument of the second-order function symbol.

To extend the semantics of  $\Sigma\text{SO}(\underline{\Lambda})$  to the case of formula  $f(x_1, \dots, x_k)$ , we say that  $F$  is a *first-order function assignment* for  $\mathcal{A}$ , if  $F(f) : A^k \rightarrow \mathcal{P}(A^*)$ . In the case of formula  $f(X)$ , we say that  $F$  is a *second-order function assignment* for  $\mathcal{A}$ , if  $F(f) : \mathcal{R}_k \rightarrow \mathcal{P}(K^*)$ , where  $K$  can be either  $A$  or  $\bigcup_{i \in \mathbb{N}} \mathcal{R}_i$ . We define  $\mathcal{FOF}$  to be the set of functions  $h : A^k \rightarrow \mathcal{P}(A^*)$ ,  $\mathcal{SOF}$  the set of functions  $h : \mathcal{R}_k \rightarrow \mathcal{P}(A^*)$ , and  $\mathcal{RSOF}$  the set of functions  $h : \mathcal{R}_k \rightarrow \mathcal{P}((\bigcup_{i \in \mathbb{N}} \mathcal{R}_i)^*)$ .

Given  $v$  and  $V$ , we define  $\text{Expl}[f(\vec{x})](\mathcal{A}, v, V, F) := F(f)(v(\vec{x}))$  and  $\llbracket f(\vec{x}) \rrbracket(\mathcal{A}, v, V, F) := |F(f)(v(\vec{x}))|$ . The semantics of  $f(X)$  are defined in an analogous way. Now we can add to the syntax of  $\Sigma\text{SO}(\underline{\Lambda})$ , formulae of the form  $[\text{lfp}_f \beta](\vec{x})$  (resp.  $[\text{lfp}_f \beta](X)$ ), where  $\beta$  is a (resp. arity-consistent)  $\Sigma\text{SO}(\underline{\Lambda})$  formula equipped with a first-order (resp. second-order) function symbol  $f$ . To define the semantics of  $[\text{lfp}_f \beta](\vec{x})$ , we first define two lattices. The first lattice is  $(\mathcal{P}(A^*), \subseteq)$ , i.e. it contains all sets of strings over  $A$ . The bottom element is  $\emptyset$  and the top element is the set  $A^*$ . The second lattice is  $(\mathcal{FOF}, \leq_F)$ : for  $g, h \in \mathcal{FOF}$ ,  $g \leq_F h$  iff  $g(\vec{x}) \subseteq h(\vec{x})$ , for every  $\vec{x}$ . The bottom element is  $g_0$  which takes the value  $\emptyset$  for every  $\vec{x}$ , and the top element is  $g_{max}$ , which is equal to  $A^*$  for every  $\vec{x}$ . For an infinite increasing sequence of functions  $h_1 \leq_F h_2 \leq_F h_3 \leq_F \dots$  from  $\mathcal{FOF}$ , we define  $\lim_{n \rightarrow +\infty} h_n := h$ , where for every  $x \in A^k$ ,  $h(x) = \bigcup_{n \in \mathbb{N}} h_n(x)$ .

We interpret  $\beta(\vec{x}, f)$  as an operator  $T_\beta$  on  $\mathcal{FOF}$ . For every  $h \in \mathcal{FOF}$  and  $\vec{a} \in A^k$ ,  $T_\beta(h)(\vec{a}) = \text{Expl}[\beta(\vec{x}, f)](\mathcal{A}, v, V, F)$ , where  $v$  is a first-order assignment for  $\mathcal{A}$  such that  $v(\vec{x}) = \vec{a}$  and  $F$  is a first-order function assignment for  $\mathcal{A}$  such that  $F(f) = h$ . The following propositions state that  $T_\beta$  is monotone on  $(\mathcal{FOF}, \leq_F)$ .

► **Proposition 3.5.** *Let  $f$  be a first-order function symbol with  $\text{arity}(f) = k$  and  $\beta$  be a formula over  $\sigma$  defined by grammar (3), such that if  $\beta$  contains a function symbol, then this function symbol is  $f$ . Let also  $\mathcal{A}$  be a finite ordered structure over  $\sigma$ ,  $h, g : A^k \rightarrow \mathcal{P}(A^*)$  and*

$H, G$  be function assignments such that  $H(f) = h$  and  $G(f) = g$ . If  $h \leq_F g$ , then for every first- and second-order assignments  $v$  and  $V$ , respectively:

$$\text{Expl}[\beta](\mathcal{A}, v, V, H) \subseteq \text{Expl}[\beta](\mathcal{A}, v, V, G).$$

► **Proposition 3.6.** For every formula  $[\text{lfp}_f \beta](\vec{x})$ , where  $\beta$  is in  $\Sigma\text{SO}(\underline{A})$  equipped with a first-order function symbol, operator  $T_\beta$  is monotone on the complete lattice  $(\mathcal{FOF}, \leq_F)$ . In other words, for every  $h, g \in \mathcal{FOF}$ , if  $h \leq_F g$ , then  $T_\beta(h) \leq_F T_\beta(g)$ .

Thus, by the Knaster–Tarski theorem,  $T_\beta$  has a least fixed point. To compute the least fixed point of  $T_\beta$ , let us consider the sequence of functions  $\{h_i\}_{i \in \mathbb{N}}$ ,  $h_i : A^k \rightarrow \mathcal{P}(A^*)$ , where  $h_0(\vec{a}) = \emptyset$  for every  $\vec{a} \in A^k$ , and  $h_{i+1} := T_\beta(h_i)$ , for every  $i \in \mathbb{N}$ . We define  $\text{lfp}(T_\beta) := \lim_{n \rightarrow +\infty} h_n$ . Finally,  $\text{Expl}[\text{lfp}_f \beta](\vec{x})(\mathcal{A}, v, V) := \text{lfp}(T_\beta)(v(\vec{x})) = \lim_{n \rightarrow +\infty} h_n(v(\vec{x}))$  and  $\llbracket [\text{lfp}_f \beta](\vec{x}) \rrbracket(\mathcal{A}, v, V) = |\lim_{n \rightarrow +\infty} h_n(v(\vec{x}))|$ . The semantics of  $[\text{lfp}_f \beta](X)$  are defined in a completely analogous way. Examples 4.3, 4.8, and 4.9 make clear how formulae of the form  $[\text{lfp}_f \beta](\vec{x})$  are interpreted.

The logics we define below are fragments of  $\Sigma\text{SO}(\underline{\text{SO}})$  with recursion. Given a formula  $[\text{lfp}_f \beta](\vec{x})$  or  $[\text{lfp}_f \beta](X)$  in any of them, operator  $T_\beta$  is monotone on the complete lattice  $(\mathcal{F}, \leq_F)$ , where  $\mathcal{F}$  can be  $\mathcal{FOF}$ ,  $\mathcal{SOF}$ , or  $\mathcal{RSOF}$ .

► **Remark 3.7.** The name of a logic with recursion will be of the form  $\mathbf{R}_{L_1} \Sigma_{L_2}(\mathbf{L}_3)$ , where  $L_1 \in \{\mathbf{fo}, \mathbf{so}\}$  indicates that function symbol  $f$  is over first- or second-order variables, respectively,  $L_2 \in \{\mathbf{fo}, \mathbf{so}\}$  means that quantifier  $\Sigma$  is over first- or second-order variables, respectively, and  $L_3 \in \{\mathbf{FO}, \mathbf{SO}\}$  means that  $\varphi$  in (2) is in  $L_3$ .

## 4 Logics that capture SpanL and SpanPSPACE

► **Definition 4.1.**  $\mathbf{R}_{\mathbf{fo}} \Sigma_{\mathbf{fo}}(\mathbf{FO})$  over  $\sigma$  is the set of formulae  $[\text{lfp}_f \beta](\vec{x})$ , where  $\beta$  is defined by:

$$\beta ::= \alpha \mid f(x_1, \dots, x_k) \mid (\beta + \beta) \mid (\alpha \cdot \beta) \mid \Sigma y. \beta \quad (4)$$

where  $\alpha$  is an  $X$ -free  $\Sigma\mathbf{FO}(\underline{\mathbf{FO}})$  formula over  $\sigma$ ,  $x_1, \dots, x_k, y$  are first-order variables, and  $f$  is a first-order function symbol.

► **Remark 4.2.** Notice that for a formula  $[\text{lfp}_f \beta](\vec{x}) \in \mathbf{R}_{\mathbf{fo}} \Sigma_{\mathbf{fo}}(\mathbf{FO})$ , it may be the case that  $\llbracket [\text{lfp}_f \beta](\vec{x}) \rrbracket(\mathcal{A}, v, V) = +\infty$  analogously to the fact that the computation of an NLTM may contain cycles. For the sake of simplicity, we assume that an NL-transducer  $M$  can have infinitely many accepting paths and  $\text{SpanL}$  contains functions from  $\Sigma^*$  to  $\mathbb{N} \cup \{+\infty\}$ . To be in accordance with the literature, we can adjust the syntax of  $\mathbf{R}_{\mathbf{fo}} \Sigma_{\mathbf{fo}}(\mathbf{FO})$  formulae to express the operation of the clock attached to NLTM as discussed in Remark 2.4.

Let  $N$  be an NL-transducer and  $\mathcal{A}$  be over  $\sigma$  with  $|A| = n$ . The number of different configurations of  $N$  is at most  $n^k - 1$  for some  $k \in \mathbb{N}$ . To encode them, we use  $k$ -tuples over  $A$ . To encode the output symbol, if any, that is produced at some configuration, it suffices to use two distinct elements of  $A$ , since the output alphabet is  $\Sigma = \{0, 1\}$ ; we use the minimum element and the successor of the minimum element, which are both  $\mathbf{FO}$  expressible. Below, we informally write  $\varphi(c)$  to denote  $\varphi(x)$  interpreted in  $\mathcal{A}$  where first-order variable  $x$  is assigned  $c \in A$ . Formula  $[\text{lfp}_f \text{span}_L](\vec{x})$  counts the different valid outputs of  $N$ , where  $\text{span}_L(\vec{x}, f)$  is:

$$\text{acc}(\vec{x}) + \Sigma \vec{y}. \Sigma z. (\text{output}_0(\vec{x}, \vec{y}, z) + \text{output}_1(\vec{x}, \vec{y}, z) + \text{next}_0(\vec{x}, \vec{y}) + \text{next}_1(\vec{x}, \vec{y})) \cdot f(\vec{y}).$$

■ **Algorithm 1** NLTM  $MSp_{\beta}^{sub}$ .

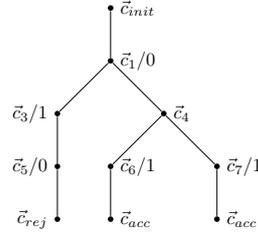
---

**Input:**  $\gamma, \mathcal{A}, v, V$ , where  $\gamma$  is a subformula of  $\beta$

- 1 **if**  $\gamma == \alpha$  has no function symbol **then** simulate the transducer from Proposition 4.5
- 2 **if**  $\gamma == f(\vec{y})$  **then** simulate  $MSp_{\beta}^{sub}(\beta, \mathcal{A}, v[v(\vec{y})/\vec{x}], V)$
- 3 **if**  $\gamma == \gamma_1 + \gamma_2$  **then**
- 4 |   non-deterministically choose  $\gamma' \in \{\gamma_1, \gamma_2\}$
- 5 |   simulate  $MSp_{\beta}^{sub}(\gamma', \mathcal{A}, v, V)$
- 6 **if**  $\gamma == \alpha \cdot \gamma'$  **then**
- 7 |   **for**  $s \in A^*$  where  $|s| \leq |\alpha|$  **do**
- 8 |   |   **if**  $s \in \text{Expl}[\alpha](\mathcal{A}, v, V)$  **then** simulate  $MSp_{\beta}^{sub}(\gamma', \mathcal{A}, v, V)$
- 9 **if**  $\gamma == \sum y.\gamma'$  **then**
- 10 |   non-deterministically choose  $a \in A$
- 11 |   simulate  $MSp_{\beta}^{sub}(\gamma', \mathcal{A}, v[a/y], V)$

---

Interpretations of  $z$  and  $\vec{x}, \vec{y}$  encode a bit of the output, and configurations of  $N$ , respectively. Formulae  $\text{next}_i(\vec{c}, \vec{c}')$ ,  $i = 0, 1$ , say that if  $N$  is in configuration  $\vec{c}$  and makes non-deterministic choice  $i$ , then it is in  $\vec{c}'$ , and no output symbol is produced. Formulae  $\text{output}_i(\vec{c}, \vec{c}', b)$ ,  $i = 0, 1$ , state that  $N$  makes choice  $i$  and so it transitions from configuration  $\vec{c}$  to  $\vec{c}'$  and writes the bit encoded by  $b$  on the next output cell. When  $N$  is in some  $\vec{c}$  that only a deterministic transition can be made, then exactly one of  $\text{next}_i(\vec{c}, \vec{c}')$ ,  $\text{output}_i(\vec{c}, \vec{c}', b)$ ,  $i = 0, 1$ , is satisfied in  $\mathcal{A}$  for a  $\vec{c}' \in A^k$  (and a  $b \in A$ ). Formula  $\text{acc}(\vec{c})$  states that  $\vec{c}$  is the accepting configuration. All aforementioned formulae can be expressed in **F0**. Note that for any  $\mathcal{A}, v$ , and  $V$ ,  $\text{Expl}[\text{lfp}_f \text{span}_{\perp}](\vec{x})(\mathcal{A}, v, V)$  is a set of strings in  $A^*$  that encode the outputs of  $N$ .



■ **Figure 1** The computation tree of a transducer  $N$  on some input  $\text{enc}(\mathcal{A})$ .  $\vec{c}/b$  represents that  $N$  enters configuration encoded by  $\vec{c}$  and writes bit  $b$  on the output tape.

► **Example 4.3.** Consider the computation tree shown in Figure 1 which corresponds to a transducer  $N$  that on input  $\text{enc}(\mathcal{A})$  has three outputs, and  $\text{span}_N(\text{enc}(\mathcal{A})) = 1$ . Let  $\mathbf{0}, \mathbf{1}$  denote the minimum and the successor of the minimum element of  $A$ , respectively. Then,

- $\text{Expl}[\text{lfp}_f \text{span}_{\perp}](\vec{x})(\mathcal{A}, v[\vec{c}_{acc}/\vec{x}]) = \{\varepsilon\}$ , and  
 $\text{Expl}[\text{lfp}_f \text{span}_{\perp}](\vec{x})(\mathcal{A}, v[\vec{c}_{rej}/\vec{x}]) = \emptyset$ ,
- $\text{Expl}[\text{lfp}_f \text{span}_{\perp}](\vec{x})(\mathcal{A}, v[\vec{c}_1/\vec{x}]) = \emptyset \cup \{\mathbf{1}\} \circ f(\vec{c}_3) \cup f(\vec{c}_4) = \{\mathbf{1}\} \circ (\mathbf{0} \circ \emptyset) \cup \{\mathbf{1}\} \circ \{\varepsilon\} = \{\mathbf{1}\}$ ,
- $\text{Expl}[\text{lfp}_f \text{span}_{\perp}](\vec{x})(\mathcal{A}, v[\vec{c}_{init}/\vec{x}]) = \emptyset \cup \{\mathbf{0}\} \circ f(\vec{c}_1) = \{\mathbf{01}\}$ .

Intuitively, the intermediate interpretation of  $\text{lfp}_f \text{span}_{\perp}(\vec{c})$  is the set of the different valid outputs  $N$  produces during its computation starting from the configuration encoded by  $\vec{c}$ .

## 7:10 Counting Computations with Formulae

► **Proposition 4.4.** *Given an NL-transducer  $N$ ,  $\text{span}_N(\text{enc}(\mathcal{A})) = \llbracket [\text{lfp}_f \text{span}_L](\vec{x}) \rrbracket(\mathcal{A}, v, V)$ , for every  $\mathcal{A}$ ,  $v$ , and  $V$ , such that  $v(\vec{x})$  encodes the starting configuration of  $N$ .*

To prove that  $\text{R}_{\text{fo}}\Sigma_{\text{fo}}(\text{FO}) \subseteq \text{SpanL}$ , first note that  $X$ -free  $\Sigma\text{FO}(\text{FO})$  formulae can be easily evaluated by NLTMs as Proposition 4.5 states.

► **Proposition 4.5.** *For every  $X$ -free  $\Sigma\text{FO}(\text{FO})$  formula  $\alpha$  over  $\sigma$ , there is an NL-transducer  $M$ , that on input  $\text{enc}(\mathcal{A}, v, V)$  has exactly one accepting run for each  $s \in \text{Expl}[\alpha](\mathcal{A}, v, V)$ , on which it outputs  $\text{enc}(s)$ , and no other accepting runs.*

► **Proposition 4.6.** *Let  $[\text{lfp}_f \beta](\vec{x})$  be an  $\text{R}_{\text{fo}}\Sigma_{\text{fo}}(\text{FO})$  formula over  $\sigma$ . There is an NL-transducer  $M_\beta$ , such that  $\text{span}_{M_\beta}(\text{enc}(\mathcal{A}, v, V)) = \llbracket [\text{lfp}_f \beta](\vec{x}) \rrbracket(\mathcal{A}, v, V)$ , for every  $\mathcal{A}$ ,  $v$  and  $V$ .*

**Proof.** Let  $[\text{lfp}_f \beta](\vec{x}) \in \text{R}_{\text{fo}}\Sigma_{\text{fo}}(\text{FO})$ . The NL-transducer  $M_\beta(\mathcal{A}, v, V)$  calls  $M\text{Sp}_\beta^{\text{sub}}(\beta, \mathcal{A}, v, V)$  from Algorithm 1. If  $\beta$  does not contain a function symbol, then  $\llbracket [\text{lfp}_f \beta](\vec{x}) \rrbracket(\mathcal{A}, v, V) = \llbracket \beta \rrbracket(\mathcal{A}, v, V)$ . By Proposition 4.5, there is an NL-transducer  $M$ , such that  $\text{span}_M(\text{enc}(\mathcal{A}, v, V)) = \llbracket \beta \rrbracket(\mathcal{A}, v, V)$ . In this case, let  $M_\beta$  be  $M$ . Similarly, for any sub-formula  $\alpha$  of  $\beta$  without function symbols,  $M_\alpha$  is the NL-transducer associated with  $\alpha$  from Proposition 4.5. ◀

► **Theorem 4.7.**  $\text{R}_{\text{fo}}\Sigma_{\text{fo}}(\text{FO}) = \text{SpanL}$  over finite ordered structures.

The following are examples of specific  $\text{SpanL}$  problems expressed in  $\text{R}_{\text{fo}}\Sigma_{\text{fo}}(\text{FO})$ .

► **Example 4.8.** Let  $\mathcal{G} = \langle V, E, \leq \rangle$  represent a directed graph with a source. Then,  $\llbracket [\text{lfp}_f \beta](x) \rrbracket(\mathcal{G}, v, V)$  is the number of sinks in the graph, where  $\beta(x, f) := \forall y \neg E(x, y) \cdot x + \Sigma y. E(x, y) \cdot f(y)$ , and  $v(x)$  is the source of the graph.

► **Example 4.9.** Let  $\mathcal{N} = \langle Q = \{q_0, \dots, q_{n-1}, \ell_0, \dots, \ell_m\}, L, E_0, E_1, \leq \rangle$  represent an NFA  $N$  over the input alphabet  $\{0, 1\}$ , together with  $1^m$ ;  $Q$  is the universe,  $L = \{\ell_0, \dots, \ell_m\}$  is a relation that distinguishes states of  $N$  from the encoding of  $1^m$ , and  $E_i$ ,  $i = 0, 1$ , is the set of  $i$ -transitions of  $N$ . Let  $\beta(x, y, f) := \text{acc}(x) + (y < \max) \cdot \Sigma x'. \Sigma y'. (y' = y + 1) \cdot (E_0(x, x') \cdot \min_0 + E_1(x, x') \cdot \min_1) \cdot f(x', y')$ , where  $\min_0$  and  $\min_1$ , and  $\max$  express the minimum, the successor of the minimum, and the maximum element of  $Q$ , respectively,  $\text{acc}(x)$  expresses that  $x$  is an accepting state of  $N$ , and  $y' = y + 1$  is defined so that  $y'$  is the successor of  $y$ . Then,  $\llbracket [\text{lfp}_f \beta](x, y) \rrbracket(\mathcal{N}, v, V)$  is the number of strings of length at most  $m$  accepted by  $N$ , where  $v(x)$  encodes the starting state of  $N$ , and  $v(y)$  encodes the minimum element of  $L$ . This problem is  $\text{SpanL}$ -complete and was defined in [2] as the *census function* of an NFA.

We now introduce the logic  $\text{R}_{\text{so}}\Sigma_{\text{so}}(\text{SO})$ , which captures  $\text{SpanPSPACE}$ .

► **Definition 4.10.**  $\text{R}_{\text{so}}\Sigma_{\text{so}}(\text{SO})$  over  $\sigma$  is the set of formulae  $[\text{lfp}_f \beta](X)$ , where  $\beta$  is defined by:

$$\beta ::= \alpha \mid f(X) \mid (\beta + \beta) \mid (\alpha \cdot \beta) \mid \Sigma y. \beta \mid \Sigma Y. \beta \quad (5)$$

where  $\alpha$  is an  $X$ -free  $\Sigma\text{SO}(\text{SO})$  formula over  $\sigma$ ,  $y$  is a first-order variable,  $X, Y$  are second-order variables, and  $f$  is a second-order function symbol.

► **Remark 4.11.** Relations  $R_1, \dots, R_m$  on  $A$  with  $\text{arity}(R_j) = k$ ,  $1 \leq j \leq m$ , can be encoded by one relation  $R$  on  $A$  of arity  $k + \lceil \log m \rceil$ , by defining  $R(\vec{i}, \vec{a})$  iff  $R_i(\vec{a})$ , for every  $\vec{a} \in A^k$ , where  $\vec{i}$  is the  $i$ -th smallest  $\lceil \log m \rceil$ -tuple over  $A$ . We use this observation to show that a second-order function symbol  $f$  with  $\text{arity}(f) = 1$ , suffices to capture  $\text{SpanPSPACE}$ .

► **Remark 4.12.** To avoid formulae  $[\text{lf}_f \beta](X) \in \mathbf{R}_{\text{so}}\Sigma_{\text{so}}(\mathbf{SO})$  with  $\llbracket [\text{lf}_f \beta](X) \rrbracket(\mathcal{A}, v, V) = +\infty$ , we can adjust the syntax of  $\mathbf{R}_{\text{so}}\Sigma_{\text{so}}(\mathbf{SO})$  similarly to Remark 4.2. The only difference is that now the clock imposes an exponential-time bound.

Let  $\mathcal{A}$  over  $\sigma$  with  $|A| = n$  and  $M = (\mathcal{Q}, \Sigma, \delta, q_0, q_F)$  be a non-deterministic poly-space transducer that uses  $n^d - 1$  space. Let also  $k = \max\{d, \lceil \log |\mathcal{Q}| \rceil\}$ . We can use  $k$ -tuples over  $A$ , to encode  $n^d - 1$  tape cells and  $|\mathcal{Q}|$  states. W.l.o.g. assume that  $M$  has a single tape. A configuration of  $M$  can be encoded by the tuple of  $k$ -ary relations  $\vec{C} = (T, E, P, Q)$ :  $T(\vec{c})$  iff cell  $c$  encoded by  $\vec{c}$  contains symbol 1 (tape contents),  $E(\vec{c})$  denotes that all cells greater than  $c$  contain the symbol  $\_$  (end of zeros and ones on the tape),  $P(\vec{c})$  indicates that the head is on cell  $c$  (head's position), and  $Q(\vec{c})$  means that  $N$  is in state  $q$  that is encoded by  $\vec{c}$ . As in the case of **SpanL**, a bit that  $M$  outputs at some time step is encoded using two elements of  $A$ . Formulae  $\text{Next}_i(\vec{X}, \vec{Y})$ ,  $\text{Output}_i(\vec{X}, \vec{Y}, x)$ ,  $i = 0, 1$ , and  $\text{Acc}(\vec{X})$  express similar facts for the computation of  $M$  as the respective formulae defined for **SpanL**. They can be expressed in **F0** as the formulae that describe the computation of an NPTM in the proof of Fagin's theorem [21]. By Remark 4.11, the aforementioned formulae can be replaced by first-order formulae such that a unique relation is used to encode the configuration of  $M$ . Therefore, we abuse notation and write  $\text{Next}_i(X, Y)$ ,  $\text{Output}_i(X, Y, x)$ , and  $\text{Acc}(X)$ .

► **Theorem 4.13.**  $\mathbf{R}_{\text{so}}\Sigma_{\text{so}}(\mathbf{SO}) = \text{SpanPSPACE}$  over finite ordered structures.

**Proof.** The proof of  $\mathbf{R}_{\text{so}}\Sigma_{\text{so}}(\mathbf{SO}) \subseteq \text{SpanPSPACE}$  is analogous to that of Proposition 4.6. For the inclusion  $\text{SpanPSPACE} \subseteq \mathbf{R}_{\text{so}}\Sigma_{\text{so}}(\mathbf{SO})$ , given a non-deterministic poly-space transducer  $M$ , consider the formula  $\text{span}_{\text{pspace}}(X, f) := \text{Acc}(X) + \Sigma Y. \Sigma x. (\text{Output}_0(X, Y, \underline{x}) + \text{Output}_1(X, Y, \underline{x}) + \text{Next}_0(X, Y) + \text{Next}_1(X, Y)) \cdot f(Y)$ . Then,  $\llbracket [\text{lf}_f \text{span}_{\text{pspace}}](X) \rrbracket(\mathcal{A}, v, V) = \text{span}_M(\text{enc}(\mathcal{A}))$ , for every  $\mathcal{A}, v, V$ , such that  $V(X)$  encodes the initial configuration of  $M$ . ◀

## 5 $\mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO})$ captures $\#PSPACE$

In this section, we prove that the logic  $\Sigma\mathbf{SO}(\underline{\mathbf{SO}})$  equipped with a second-order function symbol and a restricted form of recursion captures  $\#PSPACE$  over finite ordered structures. Superscript **r** in the name of the logic stands for the fact that recursion is restricted.

► **Definition 5.1.**  $\mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO})$  over  $\sigma$  is the set of formulae  $[\text{lf}_f \beta](X)$ , where  $\beta$  is defined by:

$$\beta ::= \alpha \mid (\alpha + \beta) \mid \Sigma Y. \varphi(X, \underline{Y}) \cdot f(Y) \quad (6)$$

where  $X, Y$  are second-order variables,  $\varphi$  is an **SO** formula over  $\sigma$ ,  $\alpha$  is an  $x$ -free  $\Sigma\mathbf{SO}(\underline{\mathbf{SO}})$  formula over  $\sigma$ , and  $f$  is a second-order function symbol.

► **Remark 5.2.** In the case of  $\#PSPACE$ , we can attach a clock to non-deterministic poly-space TMs, and restrict the syntax of  $\mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO})$  accordingly, as in Section 4. An alternative approach is the following: it can be proven that for every  $\beta \in \mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO})$ ,  $\llbracket \beta \rrbracket$  is in **FPSPACE** in the sense that there is a deterministic poly-space TM  $N$  such that on input  $\text{enc}(\mathcal{A}, v, V)$  outputs  $\llbracket \beta \rrbracket(\mathcal{A}, v, V)$ , if  $\llbracket \beta \rrbracket(\mathcal{A}, v, V) \in \mathbb{N}$ , and the symbol  $\perp$ , if  $\llbracket \beta \rrbracket(\mathcal{A}, v, V) = +\infty$ . By Proposition 2.5,  $\mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO}) \subseteq \#PSPACE$ , where we consider a slightly different kind of a non-deterministic poly-space TM which on input  $x$ , if  $f(x) = +\infty$ , it outputs  $\perp$  and halts, and if  $f(x) = m \in \mathbb{N}$ , it generates  $m$  accepting paths.

► **Theorem 5.3.**  $\mathbf{R}_{\text{so}}^r \Sigma_{\text{so}}(\mathbf{SO}) = \#PSPACE$  over finite ordered structures.

## 6

 $R_{\text{so}}^r \Sigma_{\text{so}}^r(\text{FO})$  captures TotP

We define a fragment of  $\Sigma\text{SO}(\text{FO})$  with recursion, which we call  $R_{\text{so}}^r \Sigma_{\text{so}}^r(\text{FO})$ . Definitions 6.1 and 6.2 will be used to restrict the use of  $\Sigma$  operator.

► **Definition 6.1.** We say that a formula  $\varphi(Y)$  syntactically defines  $Y$  if  $\varphi(Y)$  is of the form  $\forall \vec{y} Y(\vec{y}) \leftrightarrow \psi(\vec{y})$ , for some formula  $\psi$ .

► **Definition 6.2.** We say that a formula  $\varphi(X, Y)$  (a) extends  $X$  to  $Y$  if it is of the form  $\forall \vec{y} Y(\vec{y}) \leftrightarrow X(\vec{y}) \vee \psi(X, \vec{y})$ , and (b) strictly extends  $X$  to  $Y$  if it is of the form  $\forall \vec{y} (Y(\vec{y}) \leftrightarrow X(\vec{y}) \vee \psi(X, \vec{y})) \wedge \exists \vec{y} (\neg X(\vec{y}) \wedge Y(\vec{y}))$ , for some formula  $\psi$  and  $\text{arity}(X) = \text{arity}(Y)$ .

► **Notation Remark 6.3.** (a)  $Y := \varphi \cdot \alpha$  denotes  $\Sigma Y. \varphi(Y) \cdot \alpha$ , where  $\varphi$  syntactically defines  $Y$ , and (b)  $\underline{Y} := \varphi(X) \cdot f(Y)$  denotes  $\Sigma Y. \varphi(X, Y) \cdot Y \cdot f(Y)$ , where  $\varphi$  (strictly) extends  $X$  to  $Y$ .

► **Definition 6.4.**

(a) The  $\Sigma\text{SO}^r(\text{FO})$  formulae over  $\sigma$  are the  $x$ -free  $\Sigma\text{SO}(\text{FO})$  formulae with the restriction that the second-order sum operator only appears as  $Y := \varphi \cdot \alpha$ ,  $\varphi \in \text{FO}$ .

(b)  $R_{\text{so}}^r \Sigma_{\text{so}}^r(\text{FO})$  over  $\sigma$  is the set of formulae  $[\text{lfp}_f \beta](X)$ , where  $\beta$  is defined by:

$$\beta ::= \alpha \mid \underline{Y} := \psi(X) \cdot f(Y) \mid \alpha + \beta \mid \varphi \cdot \beta \mid \beta + \beta + \top \mid \varphi \cdot \beta + \neg \varphi \cdot \beta \quad (7)$$

where  $\alpha$  is a  $\Sigma\text{SO}^r(\text{FO})$  formula,  $\varphi, \psi \in \text{FO}$ ,  $\psi$  strictly extends  $X$  to  $Y$ , and  $f$  is a second-order function symbol.

To express the generic TotP problem in  $R_{\text{so}}^r \Sigma_{\text{so}}^r(\text{FO})$ , we first describe how an NPTM run can be encoded. Let  $\mathcal{A}$  be of size  $n$  and  $N = (\mathcal{Q}, \Sigma, \delta, q_0, q_F)$  be an NPTM that uses at most  $n^d - 1$  time. W.l.o.g. assume that  $N$  has a single tape. We define  $\Gamma = \Sigma \cup \{\_ \} = \{0, 1, \_ \}$ ,  $\Gamma_{\mathcal{Q}} = \Gamma \times \mathcal{Q}$ , and  $k = \max\{d, \lceil \log(3 + 3|\mathcal{Q}|) \rceil\}$ . To encode cells, time steps, and symbols in  $\Gamma \cup \Gamma_{\mathcal{Q}}$ , we use  $k$ -tuples over  $A$ . Let  $S$  be a relation of arity  $3k$ , such that, if  $\vec{r}$  represents the symbol  $\gamma \in \Gamma$ , then  $S(\vec{c}, \vec{t}, \vec{r})$  signifies that cell  $\vec{c}$  contains symbol  $\gamma$  at time step  $\vec{t}$ . If  $\vec{r}$  represents the symbol-state pair  $(\gamma, q) \in \Gamma_{\mathcal{Q}}$ , then  $S(\vec{c}, \vec{t}, \vec{r})$  signifies that  $\vec{c}$  contains symbol  $\gamma$ , the head is at cell  $\vec{c}$ , and  $N$  is in state  $q$  at time step  $\vec{t}$ . We use the FO expressible formulae  $\vec{x} + 1$  and  $\min$  to describe the successor of  $\vec{x}$  and the minimum  $k$ -tuple, respectively.

We say that a relation  $S$  of arity  $3k$  on  $\mathcal{A}$  describes a partial run  $c_0 c_1 \dots c_m$  of  $N$ , when (a) there is some  $\vec{t} \in A^k$ , such that for every  $\vec{t}' \leq \vec{t}$ , there are  $\vec{c}, \vec{r} \in A^k$ , such that  $S(\vec{c}, \vec{t}', \vec{r})$ , and for every  $\vec{t}' > \vec{t}$  and  $\vec{c}, \vec{r} \in A^k$ , not  $S(\vec{c}, \vec{t}', \vec{r})$ , (b)  $S(-, \min, -)$  describes the encoding of the starting configuration  $c_0$ , and (c) if  $S(-, \vec{t}, -)$  describes the encoding of  $c_i$ , then  $S(-, \vec{t} + 1, -)$  either describes the encoding of  $c_{i+1}$  or is empty. We say that formula  $\varphi(\vec{c}, \vec{t}, \vec{r})$  describes a partial run  $c_0 c_1 \dots c_m$ , when  $\varphi$  defines in  $\mathcal{A}$  a relation that does so. We use the standard notion of definability, where  $\varphi(\vec{x})$  defines  $R$  in  $\mathcal{A}$ , if for every  $\vec{a} \in A^k$ ,  $R(\vec{a})$  iff  $\mathcal{A}, v[\vec{a}/\vec{x}] \models \varphi(\vec{x})$ . For example, let  $S_0$  be a relation of arity  $3k$  that describes the beginning of a run by  $N$  on  $\text{enc}(\mathcal{A})$ .  $S_0$  can be defined in FO by  $\vec{y} = \min \wedge \varphi_{c_0}(\vec{x}, \vec{z})$ , where  $\varphi_{c_0}$  encodes the starting configuration, as, for instance, in [21].

Below we define formula  $\text{tot}(X, f)$ , the least fixed point of which applied on  $S_0$  is equal to the number of branchings of  $N$  on input  $\text{enc}(\mathcal{A})$ :

$$\text{branch}(X) \left( \sum_{i=0,1} \underline{Y} := \text{ndet}_i(X) \cdot f(Y) + \top \right) + \neg \text{branch}(X) (\text{nfinal}(X) \cdot \underline{Y} := \text{det}(X) \cdot f(Y)).$$

Let  $X$  be interpreted as a relation  $S_p$  that describes a partial run  $c_0 \dots c_m$  of  $N$ . Formula  $\text{branch}$  checks whether the current configuration  $c_m$  creates a branching. Formulae  $\text{ndet}_i$ ,  $i = 0, 1$ , and  $\text{det}$  extend  $S_p$  to a relation  $S_{\text{new}}$ , that describes the run  $c_0 \dots c_m c_{m+1}$ , where

$c_{m+1}$  is the configuration that  $N$  reaches from  $c_m$  by making non-deterministic choice  $i$  or a deterministic transition, respectively. The evaluation continues recursively on  $S_{new}$ . Finally, if  $c_m$  is a configuration where  $N$  halts,  $\mathbf{nfinal}$  becomes false and recursion stops. Moreover,  $\mathbf{ndet}_i(X, Y)$ ,  $i = 0, 1$ , and  $\mathbf{det}(X, Y)$  are  $\mathbf{FO}$  formulae that strictly extend  $X$  to  $Y$ . As a result, there is a bijection between the strings in  $\text{Expl}[\llbracket \text{lf}_f \text{tot} \rrbracket(X)](\mathcal{A}, v, V)$  and branchings of  $N(\text{enc}(\mathcal{A}))$ . Assume that  $c_m$  is a configuration that is not the initial configuration  $c_0$  and leads to a non-deterministic choice. Then,  $c_m$  can be mapped to a string  $S_1 \circ \dots \circ S_i \in (\mathcal{R}_{3k})^*$  in  $\text{Expl}[\llbracket \text{lf}_f \text{tot} \rrbracket(X)](\mathcal{A}, v, V)$ , where  $S_j$  extends  $S_{j-1}$ , for every  $2 \leq j \leq i$ , and  $S_i$  describes  $c_0 \dots c_m$ . If  $c_0$  leads to a non-deterministic choice, it is mapped to string  $\varepsilon$ .

► **Proposition 6.5.** *Given an NPTM  $N$ ,  $\llbracket \llbracket \text{lf}_f \text{tot} \rrbracket(X) \rrbracket(\mathcal{A}, v, V) = \#(\text{branchings of } N(\text{enc}(\mathcal{A})))$ , where  $V(X)$  encodes the initial configuration of  $N$ .*

The specific form of any  $\llbracket \text{lf}_f \beta \rrbracket(X) \in \mathbf{R}_{\mathbf{so}}^r \Sigma_{\mathbf{so}}^r(\mathbf{FO})$  guarantees that there is an NPTM that generates a number of paths equal to  $\llbracket \llbracket \text{lf}_f \beta \rrbracket(X) \rrbracket(\mathcal{A}, v, V) + 1$ .

► **Theorem 6.6.**  $\mathbf{R}_{\mathbf{so}}^r \Sigma_{\mathbf{so}}^r(\text{LFP}) = \text{TotP}$  over finite ordered structures.

## 7 Conclusions and open questions

Inspired by the two-step semantics developed in the context of weighted logics, we introduced two-step semantics that enriches the existing framework of quantitative logics, i.e. logics for expressing counting problems. We provided logical characterisations of  $\text{SpanL}$  and  $\text{TotP}$ , answering an open question of [5]. Furthermore, we determined logics that capture  $\text{SpanPSPACE}$  and  $\text{FPSPACE}$ . Compared to the other classes, the logic that captures  $\text{TotP}$  was defined in a more complicated way that is related to the properties of  $\text{TotP}$  problems: recursion of the logic expresses self-reducibility and the restricted form of the recursion captures the easy-decision property. It is worth investigating whether  $\text{TotP}$  is captured by a simpler, more elegant logic. The intermediate semantics can express sets of computation paths of TMs, different valid outputs of transducers, or solutions to computational problems. In particular, in the case of  $\text{SpanL}$  and  $\text{SpanPSPACE}$ , union and concatenation of sets are more suitable than addition and multiplication of  $\mathbf{QS0}$ ; when the union (resp. concatenation) of two sets of strings is computed, identical outputs will contribute one string to the resulting set. In general, using the intermediate semantics, it becomes possible to keep track of paths, outputs, and solutions, apply operations on them, and then count them. Another difference between our logics and quantitative logics from [5], is that in [5], only first-order function symbols were considered and interpreted as functions  $h : A^k \rightarrow \mathbb{N}$ . Then, the respective second lattice  $(\mathcal{F}, \leq_F)$  is not complete and the least fixed point was defined by considering the supports of functions in  $\mathcal{F}$  [5, Section 6]. By defining here, functions whose values are sets of strings, the lattice  $(\mathcal{F}, \leq_F)$ , where  $\mathcal{F}$  is one of  $\mathcal{FOF}$ ,  $\mathcal{SOF}$ , or  $\mathcal{RSOF}$ , becomes complete, and the definition of the least fixed point is straightforward.

The two-step semantics we propose in this work is noteworthy for reasons beyond its primary objective. For instance, by specifying the concrete semantics such that any non-empty set maps to 1 and the empty set to 0, our results yield least-fixed-point logical characterisations of  $\text{NL}$  and  $\text{PSPACE}$ , the decision variants of  $\text{SpanL}$  and  $\text{FPSPACE}$ , respectively. It is known that these two classes are captured by  $\mathbf{FO}$  and  $\mathbf{S0}$ , equipped with the transitive closure operator, respectively [21]. Our logics combine the least fixed point with quite natural syntactic definitions, without resorting to different fixed-point operators for each logic.

We believe that the logical characterisation of  $\text{SpanL}$  can yield more direct ways to approximate its problems.  $\mathbf{R}_{\mathbf{fo}} \Sigma_{\mathbf{fo}}(\mathbf{FO})$  formulae bear some resemblance to regular grammars, (or, equivalently, to NFAs), since the syntax of the logic, at each recursive call, concatenates

a string of fixed length from the left with  $f(\vec{x})$ . An interesting question is whether one can adjust the fpras for #NFA and apply it directly to the syntax of  $R_{\Gamma_0}\Sigma_{\Gamma_0}(\mathbf{FO})$ , giving an fpras metatheorem for the logic. Moreover, it is only natural to investigate the class that results from allowing arbitrary concatenations of recursive calls, and to expect a natural connection to context-free languages. Note that the problem of counting the strings of a specific length accepted by a context-free grammar admits a quasi-polynomial randomized approximation algorithm [19] and it is open whether it has an fpras.

Another interesting question remains the logical characterisation of a class for which computing the permanent of a matrix is complete under parsimonious reductions. This was the first problem shown in [32] to be #P-complete under Turing reductions, and it has an fpras [22]. Therefore, such a result would provide a new subclass of FPRAS and refine the complexity of the well-studied PERMANENT problem.

---

## References

- 1 Antonis Achilleos and Mathias Ruggaard Pedersen. Axiomatizations and computability of weighted monadic second-order logic. In *Proc. of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470615.
- 2 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3–30, 1993. doi:10.1016/0304-3975(93)90252-0.
- 3 Antonis Antonopoulos, Eleni Bakali, Aggeliki Chalki, Aris Pagourtzis, Petros Pantavos, and Stathis Zachos. Completeness, approximability and exponential time results for counting problems with easy decision version. *Theoretical Computer Science*, 915:55–73, 2022. doi:10.1016/j.tcs.2022.02.030.
- 4 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. *SIGMOD Record*, 49(1):52–59, 2020. doi:10.1145/3422648.3422661.
- 5 Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science*, 16(1), 2020. doi:10.23638/LMCS-16(1:9)2020.
- 6 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proc. of the 33rd Annual Symposium on Foundations of Computer Science, FOCS 1992*, pages 14–23. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267823.
- 7 Albert Atserias, Anuj Dawar, and Joanna Ochremiak. On the power of symmetric linear programs. *Journal of the ACM*, 68(4):26:1–26:35, 2021. doi:10.1145/3456297.
- 8 Eleni Bakali, Aggeliki Chalki, and Aris Pagourtzis. Characterizations and approximability of hard counting classes below #P. In *Proc. of the 16th International Conference on Theory and Applications of Models of Computation, TAMC 2020*, volume 12337 of *Lecture Notes in Computer Science*, pages 251–262, 2020. doi:10.1007/978-3-030-59267-7\_22.
- 9 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi:10.1002/malq.19600060105.
- 10 Kevin J. Compton and Erich Grädel. Logical definability of counting functions. *Journal of Computer and System Sciences*, 53(2):283–297, 1996. doi:10.1006/jcss.1996.0069.
- 11 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1):69–86, 2007. doi:10.1016/j.tcs.2007.02.055.
- 12 Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #P functions: A new perspective. *Journal of Computer and System Sciences*, 116:40–54, 2021. doi:10.1016/j.jcss.2020.04.002.
- 13 Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

- 14 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for boolean  $\#CSP$ . *Journal of Computer and System Sciences*, 76(3-4):267–277, 2010. doi: 10.1016/j.jcss.2009.08.003.
- 15 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1962. doi:10.2307/2270940.
- 16 Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972. URL: <https://books.google.is/books?id=DeLuAAAAAAAJ>.
- 17 Ronald Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974. URL: [http://www.researchgate.net/publication/242608657\\_Generalized\\_first-order\\_spectra\\_and\\_polynomial.\\_time\\_recognizable\\_sets](http://www.researchgate.net/publication/242608657_Generalized_first-order_spectra_and_polynomial._time_recognizable_sets).
- 18 Paul Gastin and Benjamin Monmege. A unifying survey on weighted logics and weighted automata - core weighted logic: minimal and versatile specification of quantitative properties. *Soft Computing*, 22(4):1047–1065, 2018. doi:10.1007/s00500-015-1952-6.
- 19 Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997. doi:10.1006/inco.1997.2621.
- 20 Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! *The Journal of Symbolic Logic*, 84(1):54–87, 2019. doi:10.1017/jsl.2018.33.
- 21 Neil Immerman. *Descriptive complexity*. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 22 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, July 2004. doi:10.1145/1008731.1008738.
- 23 Richard M Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989. doi:10.1016/0196-6774(89)90038-2.
- 24 Johannes Köbler, Uwe Schöning, and Jacobo Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989. doi:10.1007/BFb0026095.
- 25 Richard E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989. doi:10.1137/0218073.
- 26 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.
- 27 Aris Pagourtzis and Stathis Zachos. The complexity of counting functions with easy decision version. In *Proc. of the 31st International Symposium on Mathematical Foundations of Computer Science 2006, MFCS 2006*, pages 741–752. Springer, 2006. doi:10.1007/11821069\_64.
- 28 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. URL: <https://books.google.is/books?id=JogZAQAIAAJ>.
- 29 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi: 10.1016/0022-0000(91)90023-X.
- 30 Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of  $\#P$  functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995. doi: 10.1006/jcss.1995.1039.
- 31 Boris A. Trakhtenbrot. Finite automata and the logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961.
- 32 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.
- 33 Nils Vortmeier and Thomas Zeume. Dynamic complexity of parity exists queries. *Logical Methods in Computer Science*, 17(4), 2021. doi:10.46298/lmcs-17(4:9)2021.



# Recognizing $H$ -Graphs – Beyond Circular-Arc Graphs

Deniz Ağaoğlu Çağırıcı ✉

Faculty of Informatics, Masaryk University,  
Brno, Czech Republic

Jan Derbisz ✉

Theoretical Computer Science Department,  
Faculty of Mathematics and Computer Science,  
Jagiellonian University, Kraków, Poland  
Doctoral School of Exact and Natural Sciences,  
Jagiellonian University, Kraków, Poland

Petr Hliněný ✉ 

Faculty of Informatics, Masaryk University,  
Brno, Czech Republic

Tomasz Krawczyk ✉ 

Theoretical Computer Science Department,  
Faculty of Mathematics and Computer Science,  
Jagiellonian University, Kraków, Poland

Onur Çağırıcı ✉

Toronto Metropolitan University, Canada

Tim A. Hartmann ✉ 

CISPA Helmholtz Center for Information Security,  
Saarbrücken, Germany

Jan Kratochvíl ✉

Department of Applied Mathematics, Faculty of  
Mathematics and Physics, Charles University,  
Prague, Czech Republic

Peter Zeman ✉

Technical University of Denmark,  
Lyngby, Denmark

---

## Abstract

In 1992 Biró, Hujter and Tuza introduced, for every fixed connected graph  $H$ , the class of  $H$ -graphs, defined as the intersection graphs of connected subgraphs of some subdivision of  $H$ . Such classes of graphs are related to many known graph classes: for example,  $K_2$ -graphs coincide with interval graphs,  $K_3$ -graphs with circular-arc graphs, the union of  $T$ -graphs, where  $T$  ranges over all trees, coincides with chordal graphs. Recently, quite a lot of research has been devoted to understanding the tractability border for various computational problems, such as recognition or isomorphism testing, in classes of  $H$ -graphs for different graphs  $H$ .

In this work we undertake this research topic, focusing on the recognition problem. Chaplick, Töpfer, Voborník, and Zeman showed an XP-algorithm testing whether a given graph is a  $T$ -graph, where the parameter is the size of the tree  $T$ . In particular, for every fixed tree  $T$  the recognition of  $T$ -graphs can be solved in polynomial time. Tucker showed a polynomial time algorithm recognizing  $K_3$ -graphs (circular-arc graphs). On the other hand, Chaplick et al. showed also that for every fixed graph  $H$  containing two distinct cycles sharing an edge, the recognition of  $H$ -graphs is NP-hard.

The main two results of this work narrow the gap between the NP-hard and P cases of  $H$ -graph recognition. First, we show that the recognition of  $H$ -graphs is NP-hard when  $H$  contains two distinct cycles. On the other hand, we show a polynomial-time algorithm recognizing  $L$ -graphs, where  $L$  is a graph containing a cycle and an edge attached to it (which we call *lollipop graphs*). Our work leaves open the recognition problems of  $M$ -graphs for every unicyclic graph  $M$  different from a cycle and a lollipop.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** H-graphs, Intersection Graphs, Helly Property

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.8

**Related Version** *Full Version*: <https://arxiv.org/abs/2212.05433>

**Funding** *Deniz Ağaoğlu Çağırıcı*: Research of this author is supported by the Czech Science Foundation project no. 20-04567S.

*Jan Derbisz*: Research of this author was partially funded by Polish National Science Center (NCN) grant 2021/41/N/ST6/03671 and by the Priority Research Area SciMat under the program Excellence Initiative Research University at the Jagiellonian University in Kraków.



© Deniz Ağaoğlu Çağırıcı, Onur Çağırıcı, Jan Derbisz, Tim A. Hartmann, Petr Hliněný, Jan Kratochvíl, Tomasz Krawczyk, and Peter Zeman;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 8; pp. 8:1–8:14

Leibniz International Proceedings in Informatics



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

*Petr Hliněný:* Research of this author is supported by the Czech Science Foundation project no. 20-04567S.

*Peter Zeman:* Research of this author is supported by the Carlsberg Foundation Young Researcher Fellowship CF21-0682 - “Quantum Graph Theory”.

## 1 Introduction

One of the most important and most widely studied types of graph representations is an *intersection model*, in which the vertices are represented by sets and the edges by the pairs of intersecting sets. Due to practical and theoretical applications, intersection graphs of simple geometric objects are among the most intensively studied. In this paper, we consider a special kind of intersection graphs, called  *$H$ -graphs*, introduced by Biró, Hujter and Tuza [5]. Since  *$H$ -graphs* generalize many known geometric intersection graph classes, they form a good background that allows to study basic computational problems in some systematic way. We first define  *$H$ -graphs* formally using the terminology we adapt throughout the paper.

Let  $H$  be a connected graph. An  *$H$ -model* of a graph  $G$  is a pair  $(H^\phi, \phi)$ , where  $H^\phi$  is a subdivision of  $H$  and  $\phi$  is a mapping from  $V(G)$  to the subsets of  $V(H^\phi)$ , such that:

- for every  $v \in V(G)$ , the subgraph of  $H^\phi$  induced by the set  $\phi(v)$  is connected,
- for every distinct  $u, v \in V(G)$  we have  $uv \in E(G)$  iff  $\phi(u) \cap \phi(v) \neq \emptyset$ .

A graph  $G$  is an  *$H$ -graph* if  $G$  admits an  *$H$ -model*. In particular, every graph  $G$  is an  *$H$ -graph* for some graph  $H$ , e.g., for  $H = G$ .

Many known geometric intersection graph classes are  *$H$ -graphs* for an appropriately chosen graph  $H$  or are  *$\mathcal{H}$ -graphs* for some simpler class of graphs  $\mathcal{H}$ , where by  *$\mathcal{H}$ -graphs* we mean the union of the classes of  *$H$ -graphs* over  $H \in \mathcal{H}$ :

- $K_2$ -graphs coincide with the class of *interval graphs*, which are defined as the intersection graphs of intervals on the line,
- $K_3$ -graphs coincide with the class of *circular-arc graphs*, which are defined as the intersection graphs of arcs of a fixed circle,
- $\mathcal{T}$ -graphs, where  $\mathcal{T}$  contains all trees, coincide with the class of *chordal graphs*, which are defined as graphs containing no induced cycles of size  $\geq 4$  [14],
- $\mathcal{P}$ -graphs, where  $\mathcal{P}$  contains all planar graphs, coincide with the class of *string graphs*, which are defined as the intersection graphs of curves in the plane.

The recent research on  *$H$ -graphs*, initiated by Chaplick et al. [10], aims to generalize efficient optimization algorithms from simple classes of graphs on wider families of  *$H$ -graphs*, as well as to determine the boundary of “polynomial tractability” for such computational problems as recognition or isomorphism testing. Here we aim for efficient *parameterized algorithms*, whose running time depends on the size  $n$  of the input graph and the parameter  $|H|$ , where  $|H|$  is the size of the graph  $H$ . First, we search for algorithms that work in polynomial time in  $n$  and  $|H|$ , then for FPT *algorithms* working in time  $f(|H|)n^{O(1)}$  for some computable function  $f$ , and finally for XP *algorithms* working in time  $O(n^{f(|H|)})$  for some computable function  $f$ . Various NP-complete problems on  *$H$ -graphs* were studied in the parameterized setting and shown to admit FPT and XP algorithms, e.g., [1, 2, 4, 8, 10, 11, 13, 16]. Some recent research is also focused on studying the combinatorial properties of  *$H$ -graphs*, which can be later used to construct efficient algorithms in these classes of graphs (see e.g. [11, 13]).

In this work we are focusing on the recognition problem. For a graph class  $\mathcal{G}$ , the *recognition problem* for  $\mathcal{G}$  is to decide whether an input graph  $G$  belongs to  $\mathcal{G}$ . For a graph class  $\mathcal{G}$  defined in a geometrical way, the recognition problem of  $\mathcal{G}$  usually boils down to testing whether the input graph has a representation appropriate for the class  $\mathcal{G}$ . There are known linear time recognition algorithms for interval graphs [6] and chordal graphs [21].

In the context of our research, circular-arc graphs form an important class of graphs. The recognition problem for circular-arc graphs was initially thought to be NP-hard [7], but since the 1980s we already know that it admits a polynomial time algorithm [22]. Currently, two linear-time algorithms recognizing circular-arc graph are known [17,20], but both of them are rather lengthy and non-trivial. No simple polynomial-time algorithm recognizing circular-arc graphs is known.

Although the recognition of chordal graphs takes linear time, with a tree  $T$  on the input deciding whether a graph  $G$  is a  $T$ -graph is NP-complete [18]. On the other hand, Chaplick et al. [10] gave an XP algorithm parameterized by  $|T|$  recognizing  $T$ -graphs. It is open whether the problem can be solved by an FPT algorithm (in [9] it is shown that proper  $T$ -graphs can be recognized in FPT, where a  $T$ -graph  $G$  is called *proper* if there exists a  $T$ -model  $(T_\phi, \phi)$  of  $G$  such that for no pair  $u, v \in V(G)$  we have  $\phi(u) \subseteq \phi(v)$ ). Moreover, Chaplick et al. [10] showed that recognition of  $H$ -graphs is NP-complete if  $H$  contains a diamond (a cycle on four vertices with a chord) as a minor [10]. That is, recognition of  $H$ -graphs is NP-complete for every fixed  $H$  which contains two distinct cycles sharing an edge.

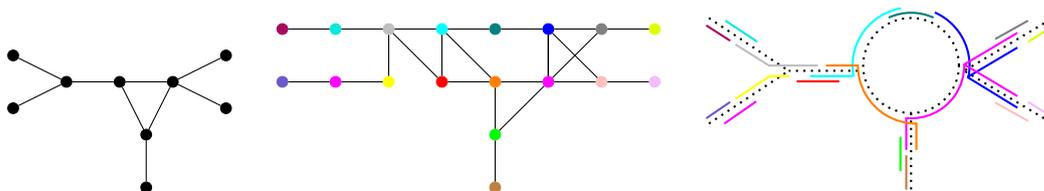
## 1.1 Our results

Our first result states the following, which extends the hardness result from [10]:

► **Theorem 1.1.** *For every fixed graph  $H$  containing two distinct cycles, the recognition of  $H$ -graphs is NP-complete.*

Theorem 1.1 raises interests in  $M$ -graphs, where  $M$  is a *unicyclic* graph (a connected graph containing exactly one cycle). In particular, we are focusing on:

- the recognition problem for the class of  $M$ -graphs, where  $M$  is any fixed graph that consists of a cycle and some trees attached to it,
- the recognition problem for the class of *medusa graphs*, which are defined as  $\mathcal{M}$ -graphs, where  $\mathcal{M}$  is the class that contains all unicyclic graphs. Note that medusa graphs extend both circular-arc graphs and chordal graphs.



■ **Figure 1.1** From left to right: a unicyclic graph  $M$ , an  $M$ -graph  $G$ , an  $M$ -model  $(M^\phi, \phi)$  of  $G$ .

An  $\mathcal{M}$ -model of a graph  $G$  is an  $M$ -model of  $G$  where  $M \in \mathcal{M}$ . Suppose  $G$  is a medusa graph and suppose  $G$  admits an  $\mathcal{M}$ -model  $(M^\phi, \phi)$  for some  $M \in \mathcal{M}$ . A clique  $C$  in  $G$  satisfies the *Helly property* in  $(M^\phi, \phi)$  if  $\bigcap_{c \in C} \phi(c) \neq \emptyset$ , and the model  $(M^\phi, \phi)$  of  $G$  satisfies the *Helly property* if every clique of  $G$  satisfies the Helly property in  $(M^\phi, \phi)$ . A medusa graph  $G$  is *Helly* if  $G$  admits an  $\mathcal{M}$ -model that satisfies the Helly property. Figure 1.1 shows a fixed unicyclic graph  $M$ , another graph  $G$  which is an  $M$ -graph, and an  $M$ -model  $(M^\phi, \phi)$  of  $G$ . Since  $(M^\phi, \phi)$  satisfies the Helly property,  $G$  is a Helly medusa graph. We show the following regarding medusa graph:

► **Theorem 1.2.**

1. *The problem of recognizing medusa graphs is NP-complete.*
2. *The problem of recognizing Helly medusa graphs is polynomial time solvable.*

Our most important (and perhaps most difficult) result concerns the class of  $L$ -graphs (which we call *lollipop graphs*), where  $L$  is a unicyclic graph that consists of a cycle with an edge attached. Note that  $L$ -graphs extend the class of circular-arc graphs.

► **Theorem 1.3.** *The problem of recognizing  $L$ -graphs is polynomial time solvable.*

Our research reveals connections between problems we consider and certain problems related to the Helly property studied in the class of circular-arc graphs. In particular, we introduce the *Helly Cliques* problem, in which for a given circular-arc graph  $G$  and its cliques  $C_1, \dots, C_k$  we need to decide whether  $G$  has a circular-arc model in which all the cliques  $C_1, \dots, C_k$  satisfy the Helly property. We show that the recognition of medusa graphs is polynomial time equivalent to the Helly Cliques problem. We refer to [3, 12] for two different proofs that the Helly Cliques problem is NP-complete. Also, to devise a polynomial algorithm recognizing  $L$ -graphs, we exploit an FPT algorithm for the Helly Cliques problem (for  $k = 1$ ) devised in [12].

## 2 Preliminaries

We refer to the full version of this paper for the full version of this section.

### 2.1 Graphs and posets

All graphs considered in this paper are *simple*, that is, they have no multiedges and no loops. We denote a complete graph and a cycle on  $n$  vertices by  $K_n$  and  $C_n$ , respectively. A *hole* in a graph is an induced cycle on at least four vertices.

A *unicyclic* graph is a connected graph that has exactly one cycle. For a unicyclic graph  $M$ , we denote by  $M_O$  the set of vertices of the unique cycle of  $M$ .

We assume that the reader has some basic knowledge on partially ordered sets (*posets*).

### 2.2 $M$ -graphs

Suppose  $M$  is a fixed unicyclic graph. Let  $(M^\phi, \phi)$  be an  $M$ -model of a graph  $G$ . If the subdivision  $M^\phi$  of  $M$  is not relevant for our considerations, we denote the model  $(M^\phi, \phi)$  simply by  $(M, \phi)$  or even by  $\phi$  (if  $M$  is clear from the context). In this case we treat  $(M, \phi)$  as the intersection model of  $G$  in which every set  $\phi(v)$  forms an arcwise connected subset of some fixed plane drawing of the unicyclic graph  $M$ . Then  $M_O$  is the part of the drawing which contains the points of the drawing corresponding to the vertices and contained in the curves representing the edges from the unique cycle of  $M$ .

Let  $M$  be a unicyclic graph. We say that a graph  $G$  is a *saturated  $M$ -graph* if  $G$  has an  $M$ -model and has no  $M^*$ -model for any proper minor  $M^*$  of  $M$ .

► **Observation 2.1.** *Suppose  $M^*$  is a minor of  $M$ . If  $G$  has an  $M^*$ -model, then  $G$  has an  $M$ -model.*

### 2.3 Interval and Circular-Arc Graphs

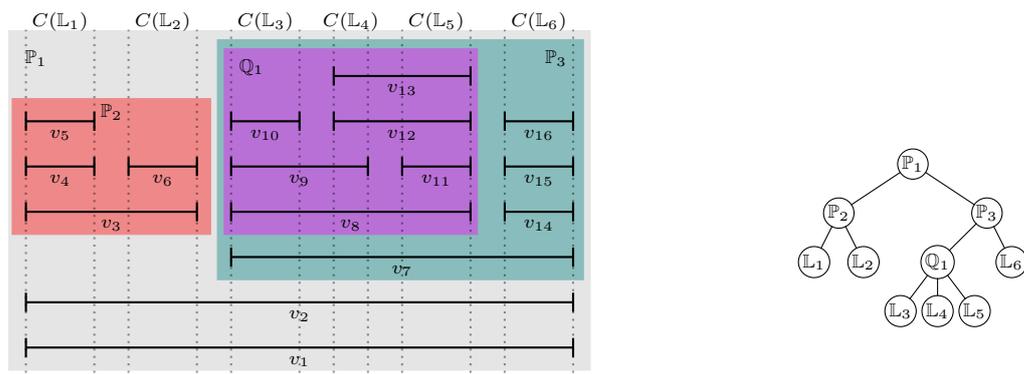
We assume that the reader has some basic knowledge of interval and circular-arc graphs. In our context we can define them as  $K_2$ -graphs and  $K_3$ -graphs.

Given an interval graph  $H$  and an interval model  $\phi$  of  $H$ , for every  $x \in \mathbb{R}$  we denote by  $C(x)$  the set  $\{v \in V(H) : x \in \phi(v)\}$ . A *sector*  $S$  of  $\phi$  is a maximal interval in  $\mathbb{R}$  such that  $C(x) = C(y)$  holds for every  $x, y \in S$ . Given a sector  $S$  of  $\phi$ , the *clique set*  $C(S)$  of  $S$  is

equal to  $C(x)$ , where  $x$  is any point inside  $S$ . Clearly, every two sectors of  $\phi$  are disjoint and the union of all sectors of  $\phi$  covers  $\mathbb{R}$  (note that  $\phi$  has at least two sectors  $S$  such that  $C(S) = \emptyset$ ). We say that a sector  $S$  of  $\phi$  is:

- *maximal* if  $C(S)$  is a maximal clique in  $H$ ,
- *minimal* if  $C(S) \subsetneq C(S')$  for any sector  $S'$  adjacent to  $S$ .

An interval model  $\phi$  of  $H$  is *normalized* if  $\phi$  has exactly  $2c + 1$  sectors (we count also the sectors with the empty clique set), where  $c$  is the number of maximal cliques in  $H$ . We refer to the full version of this paper for more details on the relations between normalized interval models of  $H$  and *consecutive orderings* of maximal cliques of  $H$  represented by the *PQ-tree* of  $H$  (see Figure 2.1).



(a) A normalized interval model  $\phi$  of  $H$ . Sectors of  $\phi$  are separated by dashed lines.

(b) *PQ*-tree  $T$  of  $H$  with leaf order  $L_1, L_2, L_3, L_4, L_5, L_6$  corresponding to consecutive clique ordering  $C_1, C_2, C_3, C_4, C_5, C_6$ .

■ **Figure 2.1**

Observe that we can obtain any interval model of  $H$  in the following way: first we choose a normalized model  $\phi$  of  $H$  (which is equivalent to picking a consecutive ordering of maximal cliques of  $H$ ) and then for each maximal sector  $S$  of  $\phi$  we shift (by a little) the endpoints of the intervals of  $\phi$  that lie on the borders of  $S$  (see Figure 2.2).

Since the definition of normalized models for circular-arc graphs is technical, we refer to the full version of this paper for the details. Here we only mention that in such models the relative relation between intersecting arcs (containment, covering the circle, or overlapping) depends on the relative relation between the closed neighbourhoods of the corresponding vertices of the graph.

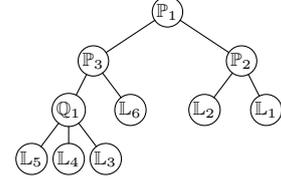
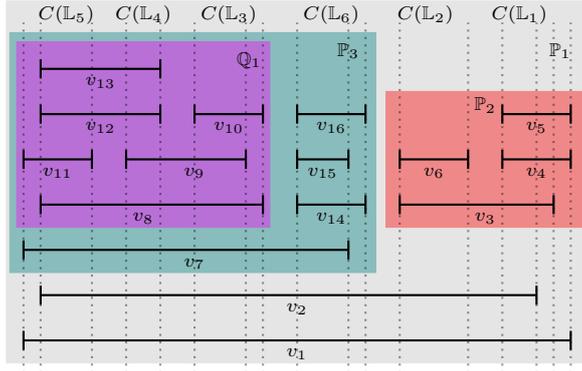
### 3 (Helly) Medusa Graphs

We refer to the full version of this paper for the full version of this section.

Recall that an  $\mathcal{M}$ -model of a graph  $G$  is an  $M$ -model of  $G$  where  $M \in \mathcal{M}$ . We introduce *normalized*  $\mathcal{M}$ -models, based on the following partition of  $V(G)$  into the *circle part*  $V_C$  and the *tree part*  $V_T$ . Start with  $V_C = \emptyset$  and repeatedly add to  $V_C$ :

- $V(C)$ , if  $C$  is a hole in  $G$ ,
- $V(P)$ , if  $P$  is an induced path in  $G$  joining two non-adjacent vertices from  $V_C$ .

Finally, let  $V_T = V \setminus V_C$ . Such a partition  $V_C \cup V_T$  of  $V(G)$  is unique and polynomial time computable. We call an  $\mathcal{M}$ -model  $(M, \phi)$  of  $G$  *normalized* if:



(a) An interval model  $\phi$  of  $H$ . Sectors of  $\phi$  are separated by dashed lines.

(b) PQ-tree  $\mathbb{T}$  of  $H$  with leaf order  $L_5, L_4, L_3, L_6, L_2, L_1$  corresponding to clique ordering  $C_5, C_4, C_3, C_6, C_2, C_1$ .

■ Figure 2.2

- $\phi(v) \cap M_O \neq \emptyset$  for every  $v \in V_C$ ,
- $\phi(v) \cap M_O = \emptyset$  for every  $v \in V_T$ ,
- $\{\phi(v) \cap M_O : v \in V_C\}$  is a normalized circular-arc model of  $G[V_C]$ .

► **Lemma 3.1.** *Every  $\mathcal{M}$ -graph  $G$  has a normalized  $\mathcal{M}$ -model  $\psi$ .*

**Sketch of the proof:** Let  $V_C \cup V_T$  be a partition of  $V(G)$  into the circle part  $V_C$  and the tree part  $V_T$  of  $G$ . Let  $(M, \psi)$  be an  $\mathcal{M}$ -model of  $G$ . The model  $(M, \psi)$  already satisfies  $\psi(v) \cap M_O \neq \emptyset$  for every  $v \in V_C$ . Let  $T_1, \dots, T_k$  be a partition of  $V_T$  into connected components of  $G[V_T]$ . Let  $N_C(T_i)$  be the neighbourhood of  $T_i$  in the cycle part of  $G$ , i.e.  $N_C(T_i) = N(T_i) \cap V_C$ . First we prove that for every  $i \in [k]$  the graph  $G[T_i \cup N_C(T_i)]$  is chordal and the set  $N_C(T_i)$  forms a clique in  $G[V_C]$ . In particular, each  $G[T_i \cup N_C(T_i)]$  has an  $F_i$ -model  $\psi_i$  for some tree  $F_i$ . Next, we note that  $\psi|_{V_C}$  restricted to  $M_O$  forms a circular arc model of  $G[V_C]$  in which each clique  $N_C(T_i)$  is Helly. We normalize this model. Finally, we obtain a normalized  $\mathcal{M}$ -model  $(M', \phi)$  of  $G$  by joining a point of  $F_i$  contained in  $\bigcap \psi_i(N_C(T_i))$  to a point of  $M_O$  contained in  $\bigcap \phi(N_C(T_i))$  and then by merging the models  $\psi$  and  $\psi_i$  for  $i \in [k]$  appropriately. ◀

This yields our main theorem characterizing medusa graphs.

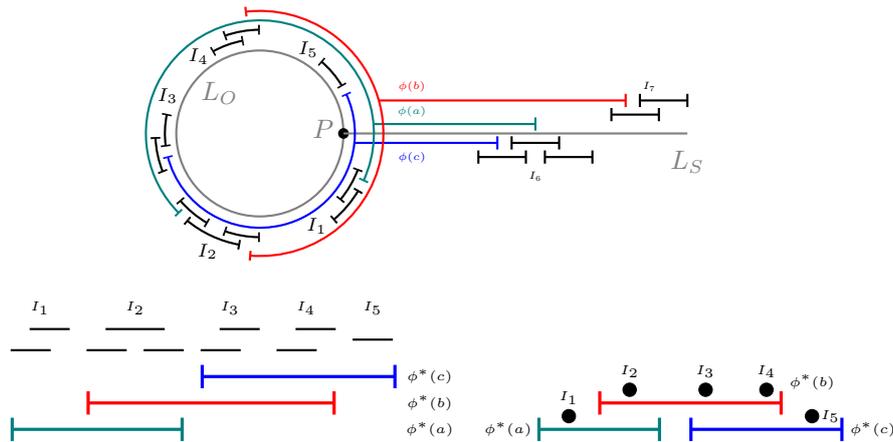
► **Theorem 3.2.** *(see the full version of this paper for the full proof) Let  $G$  be a graph, let  $V(G) = V_C \cup V_T$  be a partition of  $V(G)$  into the circle part  $V_C$  and the tree part  $V_T$  of  $G$ , and let  $T_1, \dots, T_k$  be a partition of  $V_T$  into connected components of  $G[V_T]$ . Then:*

1.  $G$  is a medusa graph if and only if  $G[V_C]$  is a circular arc graph in which for every  $i \in [k]$  the set  $N_C(T_i)$  induces a clique in  $G[V_C]$ , and  $G[V_C]$  admits a normalized circular-arc model in which every clique  $N_C(T_i)$  is Helly.
2.  $G$  is a Helly medusa graph if and only if  $G[V_C]$  is a Helly circular arc graph.

The conclusions of Theorem 3.2 bring our attention to the Helly Cliques problem.

► **Lemma 3.3.** *(see the full version of this paper for the full proof) Recognition of medusa graphs is poly-time equivalent to the Helly Cliques problem.*

We can summarize the section with the following theorem (which extends Theorem 1.2).



■ **Figure 4.1** Above:  $\{a, b, c\}$ -centered  $L$ -model  $\phi$  of  $G$ . Below left: Intervals  $\phi^*(a), \phi^*(b), \phi^*(c)$ : for  $x \in \{a, b, c\}$ ,  $\phi^*(x)$  is the shortest interval in  $L_O \setminus P$  that contains  $(L_O \setminus \phi(x))$  and every interval  $\bigcup \phi(I)$  which has a non-empty intersection with  $(L_O \setminus \phi(x))$ . Below right: schematic view of  $\phi^*$  with the components  $I_1, \dots, I_5$ . Our second key step should output  $\mathcal{H}$  containing  $H$  such that  $V(H) = \{a, b, c\}$  and  $E(H) = \{ab, bc\}$ .

► **Theorem 3.4.**

1. *The problem of recognizing medusa graphs is NP-complete.*
2. *The problem of recognizing medusa graphs parameterized by the number  $k$  of components in the tree part  $G[V_T]$  of the input graph is FPT.*
3. *The problem of recognizing Helly medusa graphs is polynomial time solvable.*

**Proof.** The statements of the theorem follow from Lemma 3.3, from the fact that the Helly Cliques problem is NP-complete [3,12] and can be solved in time  $2^{O(k \log k)} \text{poly}(n)$  [12], and from the fact that Helly circular-arc graphs recognition can be solved in linear-time [19]. ◀

**4 Lollipop Graphs**

In this section we derive a polynomial time algorithm for recognizing  $L$ -graphs, where  $L$  is the graph which consists of the clique  $K_3$  and an edge attached to one vertex of  $K_3$  ( $L$  is called a *lollipop* and  $L$ -graphs are called *lollipop graphs*). Since there are known polynomial-time algorithms recognizing  $K_{1,3}$ -graphs [10] and  $K_3$ -graphs [17, 20], we assume that an input graph  $G = (V, E)$  is not an  $L^*$ -graph for any proper minor  $L^*$  of  $L$ . Hence our goal is to test whether  $G$  is a saturated  $L$ -graph.

We fix a plane drawing of  $L$  which consists of the circle  $L_O$  and the stick  $L_S$  attached to  $L_O$  in the point  $P$  (see Figure 4.1 for an illustration). Then we treat an  $L$ -model  $\phi$  of  $G$  as the intersection model of  $G$  in which every set  $\phi(v)$  forms an arcwise connected subset of the drawing of  $L$ . We call the arcs contained in  $L_O \setminus P$  as *intervals* and we introduce *left-right* orders of the points in  $L_O \setminus P$  (consistent with the clockwise order) and in  $L_S \setminus P$ .

Let  $C$  be a clique of  $G$ . An  $L$ -model  $\phi$  of  $G$  is  $C$ -centered if  $C = \{v \in V : P \in \phi(v)\}$  and  $G$  is  $C$ -centered if  $G$  admits a  $C$ -centered  $L$ -model. For example, the model  $\phi$  shown in Figure 4.1 is  $\{a, b, c\}$ -centered.

Our approach consists of three key steps. The first step is summarized as follows.

► **Theorem 4.1.** *There is a polynomial time algorithm that, given a graph  $G$ , either decides that  $G$  is a saturated  $L$ -graph, or outputs a set of cliques  $\mathcal{C}$  of  $G$  such that, if  $G$  is a saturated  $L$ -graph then  $G$  is a  $C$ -centered  $L$ -graph for some  $C \in \mathcal{C}$ .*

Given the above theorem it remains to efficiently decide whether  $G$  is a  $C$ -centered graph for some fixed clique  $C$  of  $G$ . Let  $\mathcal{I}$  denote the set of components of  $G[V \setminus C]$ . Note that for every  $I \in \mathcal{I}$  the set  $I$  induces an interval graph in  $G$  if  $G$  is  $C$ -centered.

Suppose  $\phi$  is a  $C$ -centered model of  $G$ . To describe our second step, for every  $c \in C$  let  $\phi^*(c)$  denote the shortest interval in  $L_O \setminus P$  containing the interval  $L_O \setminus \phi(c)$  and every interval  $\bigcup \phi(I)$  whenever  $L_O \setminus \phi(c)$  intersects  $\bigcup \phi(I)$ , for  $I \in \mathcal{I}$ . Let  $H_\phi$  be an interval graph with the vertex set  $C' = \{c \in C : \phi^*(c) \neq \emptyset\}$  and with the intersection model given by the intervals  $\{\phi^*(c) : c \in C'\}$ .

Let  $C' \subseteq C$  and let  $H$  be an interval graph on the vertex set  $C'$ . A  $C$ -centered  $L$ -model  $\phi$  of  $G$  is  $(C, H)$ -centered if  $H_\phi = H$  and  $G$  is  $(C, H)$ -centered if  $G$  admits a  $(C, H)$ -centered model. For example, the model  $\phi$  from Figure 4.1 is  $(\{a, b, c\}, H)$ -centered, where  $H$  is such that  $V(H) = \{a, b, c\}$  and  $E(H) = \{ab, bc\}$ . Our second step comes down to the following.

► **Theorem 4.2.** *There is a polynomial time algorithm that, given a graph  $G$  and a clique  $C \subseteq V(G)$ , outputs a set of interval graphs  $\mathcal{H}$  such that, if  $G$  is a  $C$ -centered  $L$ -graph, then  $G$  is a  $(C, H)$ -centered  $L$ -graph for some  $H \in \mathcal{H}$ .*

Our final step, which is the most technical and probably most difficult, can be summarized as follows:

► **Theorem 4.3.** *There is a polynomial time algorithm that, given a graph  $G$ , a clique  $C \subseteq V(G)$ , and an interval graph  $H$  on a subset  $C'$  of  $C$ , decides whether  $G$  is a  $(C, H)$ -centered graph.*

The algorithm from Theorem 4.3 exploits dynamic programming along the PQ-tree of  $H$  to test whether there is a partition  $(\mathcal{J}, \mathcal{J}')$  of the components of  $\mathcal{I}$  together with a total ordering  $\prec$  of  $\mathcal{J}$  such that there is a  $(C, H)$ -centered model  $\phi$  of  $G$  that places the interval graphs of  $\mathcal{J}$  on  $L_O \setminus P$  in the order  $\prec$  and the interval graphs of  $\mathcal{J}'$  on the stick  $L_S$ .

#### 4.1 Sketch of the proof of Theorem 4.1 (the first key step)

We refer to the full version of this paper for the full proof.

Let  $G$  be a graph which is neither  $K_{1,3}$ -graph nor  $K_3$ -graph. Let  $V_C \cup V_T$  be the partition of  $V(G)$  into the circle part  $V_C$  and the tree part  $V_T$  of  $G$ , and let  $\mathcal{T}$  be the set of all maximal cliques of the chordal graphs  $G[T \cup N_C(T)]$ , where  $T$  runs over the components of  $G[V_T]$  (see Section 3). The algorithm for Theorem 4.1 works as follows. For every  $C^* \in \mathcal{T}$ :

- output  $C^*$  and denote by  $\mathcal{I}^*$  the components of  $G[V \setminus C^*]$ ,
- for every component  $I \in \mathcal{I}^*$  let  $C' = \{v \in C^* : v \text{ has a neighbour in } I\}$ ,
  - accept  $G$  as saturated  $L$ -graph if  $G[I \cup C']$  admits a circular arc model with  $C'$  Helly (we use a poly-time algorithm for the Helly Cliques problem with  $k = 1$ ) and  $G[V \setminus I]$  admits an interval model with  $C^*$  as the leftmost maximal clique,
  - if  $G[I]$  is an interval graph, then for every maximal clique  $D$  of  $G[I]$  output the clique  $C(D) = C' \cup \{v \in D : C' \subseteq N(v)\}$ .

Now we give a sketch of the proof that this algorithm is correct. Assuming  $G$  is a saturated  $L$ -graph, we first argue  $G$  admits an  $L$ -model  $\phi$  such that:

- there is a clique  $C^*$  from  $\mathcal{T}$  such that the set  $\bigcap \phi(C^*)$  is contained in  $L_S \setminus P$  and  $\bigcap \phi(C^*)$  is as close to  $P$  as possible,

- for every component  $I \in \mathcal{I}^*$  we have either  $\bigcup \phi(I) \subseteq L_O$  or  $\bigcup \phi(I) \subseteq (L_S \setminus P)$ .

As we show in the full version of this paper, such properties are satisfied by so-called *saturated*  $L$ -models of  $G$ . Given such  $\phi$ , we fall into one of the following cases:

- $P$  is not covered by  $\bigcup \phi(I)$  for any  $I \in \mathcal{I}^*$ . Then  $\phi$  can be easily turned into  $C^*$ -centered  $L$ -model of  $G$  (since the algorithm adds  $C^*$  to  $\mathcal{C}$ , Theorem 4.1 is satisfied).
- $P$  is covered by  $\bigcup \phi(I)$  for some  $I \in \mathcal{I}^*$ . The algorithm accepts  $G$  if  $I$  is the only component on  $L_O$  (which possibly induces a circular-arc graph and covers the whole circle  $L_O$ ). Otherwise, if two components from  $\mathcal{I}^*$  are represented on  $L_O$ , then all the components in  $\mathcal{I}^*$  induce interval graphs in  $G$ . In this case we argue that  $\phi$  can be turned into a  $C(D)$ -centered model, where  $D$  is a maximal clique in  $G[I]$  with  $\bigcap \phi(D)$  next to  $P$  (since the algorithm adds  $C(D)$  to  $\mathcal{C}$ , Theorem 4.1 is satisfied).

Given the previous, the algorithm for Theorem 4.1 either accepts  $G$ , or outputs  $\mathcal{C}$  of size at most  $O(n^2)$  ( $\mathcal{T}$  has size  $O(n)$  and for every  $C^* \in \mathcal{T}$  the total number of maximal cliques in the interval components of  $G[V \setminus C^*]$  is  $O(n)$ ).

## 4.2 Sketch of the proof of Theorem 4.2 (the second key step)

We refer to the full version of this paper for the full proof.

Consider a graph  $G$  and a clique  $C$  of  $G$ . Let  $\mathcal{I}$  be the set of components of  $G[V \setminus C]$ . For now, consider a  $C$ -centered model  $\phi$ . Then every  $I \in \mathcal{I}$  induces an interval graph and  $\{\phi(v) \mid v \in I\}$  is an interval model for  $G[I]$ . We partition  $I \in \mathcal{I}$  according to the intersection with  $L_O$  and  $L_S$  in the model  $\phi$ :

$$\mathcal{I}_O^\phi = \left\{ I \in \mathcal{I} : \bigcup \phi(I) \subseteq L_O \right\} \quad \text{and} \quad \mathcal{I}_S^\phi = \left\{ I \in \mathcal{I} : \bigcup \phi(I) \subseteq L_S \right\}.$$

For  $I \in \mathcal{I}$ , let  $C(I) = \{c \in C : c \text{ is not adjacent to some vertex in } I\}$ . Let  $\phi$  be a  $C$ -centered model of  $G$ . Note that the graph  $H_\phi$ , defined in Section 4, can be equivalently defined such that

$$C' = V(H_\phi) = \bigcup_{I \in \mathcal{I}_O^\phi} C(I) \quad \text{and} \quad E(H_\phi) = \{\{c, c'\} : \text{there is } I \in \mathcal{I}_O^\phi \text{ such that } c, c' \in C(I)\}.$$

Similarly, the model  $\phi^*$  of  $H_\phi$  can be equivalently defined such that for every  $c \in C'$  the set  $\phi^*(c)$  is the shortest interval that contains  $\bigcup \phi(I)$  for every  $I \in \mathcal{I}_O^\phi$  such that  $c \in C(I)$  – see Figure 4.1 for an illustration. Note that every non-minimal sector of  $\phi^*$  contains at least one component from  $\mathcal{I}_O^\phi$  and every  $I \in \mathcal{I}_O^\phi$  occupies a sector of  $\phi^*$  with the clique set  $C(I)$ .

We distinguish the interval graphs  $I \in \mathcal{I}$  as follows:

- $I \in \mathcal{I}$  is called *ambiguous* if  $G[C \cup I]$  has an interval model with  $C$  as its left-most clique (hence,  $I$  might be placed on the stick); otherwise  $I$  is called *circle*.
- An ambiguous component  $I \in \mathcal{I}$  is *simple* if  $N(u) \cap C = N(v) \cap C$  for all  $u, v \in I$ .

We denote the sets of the circle, ambiguous, ambiguous simple, and ambiguous non-simple components by  $\mathcal{I}_c$ ,  $\mathcal{I}_a$ ,  $\mathcal{I}_{as}$ , and  $\mathcal{I}_{ans}$ , respectively. Clearly,  $\mathcal{I}_c \subseteq \mathcal{I}_O^\phi$ , for every  $C$ -centered model  $\phi$ . Moreover, we show that for every  $I \in \mathcal{I}_c$  and every  $C$ -centered model  $\phi$  of  $G$  the component  $I$  occupies a maximal sector of  $\phi^*$  and no other component occupies this sector – see the circle component  $I_2$  in Figure 4.1. We additionally assume that  $C(I) \neq C(J)$  for every two simple components  $I \neq J$  since otherwise we may simply consider the input without  $J$ .

## 8:10 Recognizing $H$ -Graphs – Beyond Circular-Arc Graphs

We order  $\mathcal{I}$  based on their subset relation on  $\{C(I) : I \in \mathcal{I}\}$  and their classification into  $\mathcal{I}_a$ ,  $\mathcal{I}_{as}$ , and  $\mathcal{I}_{ans}$ . We define  $\subset_t$  for  $I, J \in \mathcal{I}$  as

$$I \subset_t J \text{ if } \begin{aligned} & (C(I) \subsetneq C(J)) \text{ or} \\ & (C(I) = C(J) \text{ and } I \in \mathcal{I}_c \text{ and } J \in \mathcal{I}_a) \text{ or} \\ & (C(I) = C(J) \text{ and } I \in \mathcal{I}_{ans} \text{ and } J \in \mathcal{I}_{as}). \end{aligned}$$

Then  $(\mathcal{I}, \subseteq_t)$  is a poset, where  $\subseteq_t$  is the reflexive closure of  $\subset_t$ .

▷ **Claim 4.4.** For every  $C$ -centered model  $\phi$  we have  $\mathcal{I}_S^\phi \subseteq \mathcal{I}_a$ , the components from  $\mathcal{I}_S^\phi$  form a chain in  $(\mathcal{I}, \subseteq_t)$ , and the left-right order of the components on  $L_S$  coincides with  $(\mathcal{I}_S^\phi, \subseteq_t)$ .

Let  $\mathcal{D} \subseteq \mathcal{I}$  be defined as follows. For every  $J \in \mathcal{I}_a$ ,

(D1) if  $C(I) \subseteq C(J)$  for some  $I \in \mathcal{I}_c$ , add  $J$  to  $\mathcal{D}$ .

(D2) if  $J \in \mathcal{I}$  is such that the downset of  $J$  has the width at least 4 in  $(\mathcal{I}, \subseteq_t)$ , add  $J$  to  $\mathcal{D}$ . We show that the components from  $\mathcal{D}$  need to be represented on the stick in any  $C$ -centered model of  $G$ .

Since every maximal sector of  $\phi^*$  contains a component from  $\mathcal{A}^\phi$  and the components from  $\mathcal{A}^\phi$  occupy maximal sectors of  $\phi^*$ , the set  $\mathcal{A}^\phi$  determines the set of all maximal cliques of  $H_\phi$ . Hence, the interval graph  $H_\phi$  is uniquely determined by the antichain  $\mathcal{A}^\phi$  containing the maximal components from  $(\mathcal{I}_O^\phi, \subseteq_t)$ . Therefore, we refer to  $H_\phi$  as to  $H(\mathcal{A}^\phi)$ .

Let  $\mathcal{I}' = \mathcal{I} \setminus \mathcal{D}$ . Let  $\mathcal{A}$  be the set of the maximal components in  $(\mathcal{I}', \subseteq_t)$ . Let  $\phi$  be a  $C$ -centered  $L$ -model of  $G$ . Since  $\mathcal{A}$  forms an antichain and  $\mathcal{I}_S^\phi$  forms a chain, their intersection contains at most one element. If  $\mathcal{A} \cap \mathcal{I}_S^\phi = \emptyset$ , then  $\mathcal{A}^\phi = \mathcal{A}$  and we output  $H(\mathcal{A})$  as a candidate. If  $|\mathcal{A} \cap \mathcal{I}_S^\phi| = \{A\}$ , then  $\mathcal{A}^\phi = (\mathcal{A} \setminus \{A\}) \cup \mathcal{A}_3$ , where  $\mathcal{A}_3$  is an antichain in the downset of  $A$ . Hence we try all of the  $O(n)$  maximal components in  $A \in \mathcal{A}$  combined with all antichains in  $DS(A)$ . Since  $A \notin \mathcal{D}$ , its downset has the width at most 3 and there are  $O(n^3)$  antichains in  $DS(A)$  which can be enumerated in polynomial time.

### 4.3 Sketch of the proof of Theorem 4.3 (the third key step)

We refer to the full version of this paper for the full proof.

First, we reduce the input instance  $G, C, H$ , so as there is no component  $I \in \mathcal{I}$  such that  $C(I)$  is the clique set of a minimal sector of an interval model of  $H$  (the set of cliques of minimal sectors of an interval model of  $H$  is independent on the model of  $H$ ).

Let  $\mathbb{T}$  be the PQ-tree of  $H$ , let  $V(\mathbb{T})$  be the nodes of  $\mathbb{T}$ , and let  $\mathbb{R}$  be the root of  $\mathbb{T}$ . For  $\mathbb{N} \in V(\mathbb{T})$ , let  $\mathcal{L}(\mathbb{N})$  denote the set of all leaves of  $\mathbb{T}$  which descend  $\mathbb{N}$  in  $\mathbb{T}$ . We set  $\mathcal{L} = \mathcal{L}(\mathbb{R})$ . For  $\mathbb{L} \in \mathcal{L}$  let  $C(\mathbb{L})$  denote the clique of  $H$  represented by  $\mathbb{L}$ . We refer to the full version of this paper for more about PQ-trees.

Our main task is to find a partition  $(\mathcal{J}, \mathcal{J}')$  of  $\mathcal{I}' = \mathcal{I} \setminus \mathcal{D}$  together with a total ordering  $\prec$  of  $\mathcal{J}$  such that there is a model  $\phi$  that places the interval graphs of  $\mathcal{J}$  on the circle  $L_O$  in the order  $\prec$  and the interval graphs of  $\mathcal{J}'$  on the stick  $L_S$ .

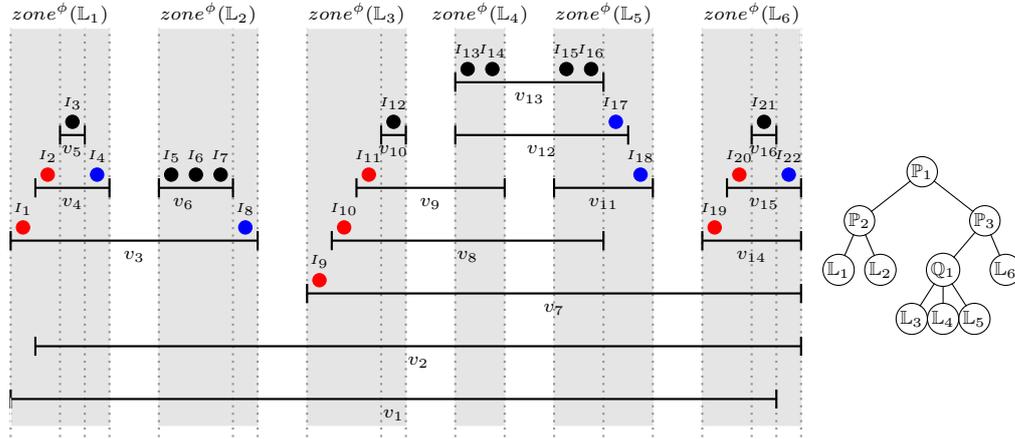
The easy part is to check whether  $\mathcal{J}' \subseteq \mathcal{I}'$  can be placed on the stick. We need to check whether  $(\mathcal{J}' \cup \mathcal{D})$  forms a chain in the poset  $(\mathcal{I}_a, \sqsubseteq)$ , with  $\sqsubseteq$  being the reflexive closure of the following binary relation  $\sqsubset$ , defined for distinct  $I, J \in \mathcal{I}_a$ :

$$I \sqsubset J \text{ if } C(I) \subseteq C^s(J), \text{ where} \\ C^s(J) = \{c \in C : c \text{ is not adjacent to every vertex in } J\}.$$

Now, let us consider how to test whether  $\mathcal{J} \subseteq \mathcal{I}'$  together with an ordering  $\prec$  allows a model that places the interval graphs of  $\mathcal{J}$  in the order of  $\prec$  on the circle. We will denote such an ordering  $(\mathcal{J}, \prec)$  as a *good order* for  $\mathbb{R}$ . Our final dynamic program then determines the

good orderings in a bottom up fashion. Hence we will define a *good order*  $(\mathcal{K}, \prec)$  with respect to some node  $\mathbb{N}$  of  $\mathbb{T}$  and with respect to some left and right *borders*,  $(B_L, B_R)$ , which are cliques in  $H$ . Since this definition is technical (see the full version of this paper), we describe some properties of good orderings that occur in  $(\mathcal{I}_O^\phi, \prec_\phi)$ , where  $\phi$  is a  $(C, H)$ -centered model of  $G$  and  $(\mathcal{I}_O^\phi, \prec_\phi)$  is the left-right order of the components from  $\mathcal{I}_O^\phi$  on  $L_O \setminus P$ .

Let  $\phi$  be a  $(C, H)$ -centered model of  $G$ . For  $\mathbb{L} \in \mathcal{L}$  let  $\text{inner}(\mathbb{L}) = \{I \in \mathcal{I}' : C(I) = C(\mathbb{L})\}$  and let  $\text{inner}_O^\phi(\mathbb{L}) = \text{inner}(\mathbb{L}) \cap \mathcal{I}_O^\phi$ , and  $\text{inner}_S^\phi(\mathbb{L}) = \text{inner}(\mathbb{L}) \cap \mathcal{I}_S^\phi$ . Additionally, assume the maximal cliques of  $H$  appear in  $\phi^*$  in the order  $C(\mathbb{L}_1), \dots, C(\mathbb{L}_n)$ . For every  $\mathbb{L}_i \in \mathcal{L}$  we define the *left zone*  $\text{zone}_L^\phi(\mathbb{L}_i)$  of  $\mathbb{L}_i$  as an interval of  $(\mathcal{I}_O^\phi, \prec_\phi)$  including the components from the sectors of  $\phi^*$  contained strictly between the maximal sector  $S(\mathbb{L}_i)$  with the clique set  $C(\mathbb{L}_i)$  and the minimal sector of  $\phi^*$  preceding  $S(\mathbb{L}_i)$  in  $\phi^*$ . We define the *right zone*  $\text{zone}_R^\phi(\mathbb{L}_i)$  of  $\mathbb{L}_i$  analogously and we set  $\text{zone}^\phi(\mathbb{L}_i) = \text{zone}_L^\phi(\mathbb{L}_i) \cup \text{inner}_O^\phi(\mathbb{L}_i) \cup \text{zone}_R^\phi(\mathbb{L}_i)$ . For a non-leaf node  $\mathbb{N}$ , let  $\text{zone}^\phi(\mathbb{N}) = \bigcup_{\mathbb{L} \in \mathcal{L}(\mathbb{N})} \text{zone}^\phi(\mathbb{L})$ . Figure 4.2 shows the zones for some example model  $\phi^*$ . Components from the sets  $\text{zone}_L^\phi(\mathbb{L}_i)$ ,  $\text{inner}_O^\phi(\mathbb{L}_i)$ , and  $\text{zone}_R^\phi(\mathbb{L}_i)$  are illustrated as red, black, and blue dots, respectively. We have, for example,  $\text{zone}_L^\phi(\mathbb{L}_3) = \{I_9, I_{10}, I_{11}\}$ ,  $\text{zone}_R^\phi(\mathbb{L}_2) = \{I_8\}$ , and  $\text{inner}_O^\phi(\mathbb{L}_2) = \{I_5, I_6, I_7\}$ . We have  $\text{zone}^\phi(\mathbb{Q}_1) = \{I_9, \dots, I_{18}\}$  as  $\mathcal{L}(\mathbb{Q}_1) = \{\mathbb{L}_3, \mathbb{L}_4, \mathbb{L}_5\}$ .



■ **Figure 4.2** Zones in  $\phi^*$ .

- Among others, we show that for every  $i \in [n]$  (below we assume  $C(\mathbb{L}_0) = C(\mathbb{L}_{n+1}) = \emptyset$ ):
- the set  $\text{zone}_L^\phi(\mathbb{L}_i)$  forms a chain in  $(\mathcal{I}_a, \sqsubseteq)$ ,  $(\text{zone}_L^\phi(\mathbb{L}_i), \sqsubseteq)$  is equal to  $(\text{zone}_L^\phi(\mathbb{L}_i), \prec_\phi)$  and  $\text{zone}_L^\phi(\mathbb{L}_i)$  respects the border  $C(\mathbb{L}_{i-1}) \cap C(\mathbb{L}_i)$ , which means  $C(\mathbb{L}_{i-1}) \cap C(\mathbb{L}_i) \sqsubseteq \text{zone}_L^\phi(\mathbb{L}_i)$ ,
- the set  $\text{zone}_R^\phi(\mathbb{L}_i)$  forms a chain in  $(\mathcal{I}_a, \sqsubseteq)$ , we have  $(\text{zone}_R^\phi(\mathbb{L}_i), \sqsubseteq)$  is equal to the reverse of  $(\text{zone}_R^\phi(\mathbb{L}_i), \prec_\phi)$  and  $\text{zone}_R^\phi(\mathbb{L}_i)$  respects the border  $C(\mathbb{L}_i) \cap C(\mathbb{L}_{i+1})$ , which means  $C(\mathbb{L}_i) \cap C(\mathbb{L}_{i+1}) \sqsubseteq \text{zone}_R^\phi(\mathbb{L}_i)$ .

The next step is to define when  $(\mathcal{K}, \prec)$  is a *good order* for  $\mathbb{N}$  and borders  $(B_L, B_R)$ . Roughly speaking,  $(\mathcal{K}, \prec)$  is defined such that it allows to derive an admissible order  $\mathbb{L}_1, \dots, \mathbb{L}_m$  of the leaves from  $\mathcal{L}(\mathbb{N})$  and to define the zones for  $\mathbb{L}_i$  in  $(\mathcal{K}, \prec)$  for all  $i \in [m]$ . Moreover, it is required that the left zone of  $\mathbb{L}_1$  respects the border  $B_L$ , the right zone of  $\mathbb{L}_m$  respects the border  $B_R$ , and the right zone of  $\mathbb{L}_i$  and the left zone of  $\mathbb{L}_{i+1}$  respect the border  $C(\mathbb{L}_i) \cap C(\mathbb{L}_{i+1})$  for  $i \in [m-1]$ . For example, in Figure 4.2  $(\text{zone}^\phi(\mathbb{L}_3), \prec_\phi)$  is a good order for  $\mathbb{L}_3$  and the borders  $(\{v_1, v_2\}, \{v_1, v_2, v_7, v_8, v_9\})$ ,  $(\text{zone}^\phi(\mathbb{Q}_1), \prec_\phi) = (I_9, \dots, I_{18})$  is a good order for  $\mathbb{Q}_1$  and for the borders  $(\{v_1, v_2\}, \{v_1, v_2, v_7\})$ . Finally, the full definition of good orders allows us to prove Theorem 4.6, which boils down the problem of searching for a  $(C, H)$ -centered model of  $G$  to the problem of testing whether there is a “good triple” for the set  $\mathcal{I}'$  and the node  $\mathbb{R}$ .

► **Definition 4.5.** Let  $\mathbb{N}$  be a node of  $\mathbb{T}$  and  $\mathcal{I}^* \subseteq \mathcal{I}'$ . A triple  $(\mathcal{J}, \mathcal{J}', \prec)$  is good for the set  $\mathcal{I}^*$ , the node  $\mathbb{N}$ , and the borders  $(B_L, B_R)$ , if:

- $\{\mathcal{J}, \mathcal{J}'\}$  is a partition of  $\mathcal{I}^*$ .
- $(\mathcal{J}, \prec)$  is a good order for the node  $\mathbb{N}$  and the borders  $(B_L, B_R)$ ,
- $\mathcal{J}' \cup \mathcal{D}$  is a chain in  $(\mathcal{I}_a, \sqsubseteq)$ .

If  $(B_L, B_R) = (\emptyset, \emptyset)$ , we simply say  $(\mathcal{J}, \mathcal{J}', \prec)$  is good for the set  $\mathcal{I}^*$  and the node  $\mathbb{N}$ .

With the notion of a good triple we obtain the following theorem that characterizes all  $(C, H)$ -centered models of  $G$ .

► **Theorem 4.6** (see the full version of this paper for the full proof). Let  $G$  be a graph.

1. For every  $(C, H)$ -centered model  $\phi$  of  $G$  the triple  $(\mathcal{I}_O^\phi, \mathcal{I}_S^\phi \setminus \mathcal{D}, \prec_\phi)$  is good for  $\mathcal{I}'$  and the node  $\mathbb{R}$ .
2. For every triple  $(\mathcal{J}, \mathcal{J}', \prec)$  that is good for  $\mathcal{I}'$  and the node  $\mathbb{R}$ , there is a  $(C, H)$ -centered model  $\phi$  of  $G$  such that  $(\mathcal{J}, \prec) = (\mathcal{I}_O^\phi, \prec_\phi)$  and  $\mathcal{J}' \cup \mathcal{D} = \mathcal{I}_S^\phi$ .

Our algorithm needs to test whether there exists a good triple for the set  $\mathcal{I}' = \text{comp}(\mathbb{R})$ . Roughly speaking, this technical part is done as follows: first, we carefully define the sets  $\text{comp}(\mathbb{N}) \subseteq \mathcal{I}'$  for every node  $\mathbb{N} \in V(\mathbb{T})$  and then we compute good triples<sup>1</sup> for every set  $\text{comp}(\mathbb{N})$  using dynamic programming over  $\mathbb{T}$ .

## 5 Butterfly-Graphs

Here we sketch an approach to proving Theorem 1.1. The main task is to show NP-hardness of recognizing butterfly-graphs, where a butterfly is the graph consisting of two  $K_3$ 's joined on one vertex.

► **Theorem 5.1.** *BUTTERFLY-GRAPH RECOGNITION is NP-complete.*

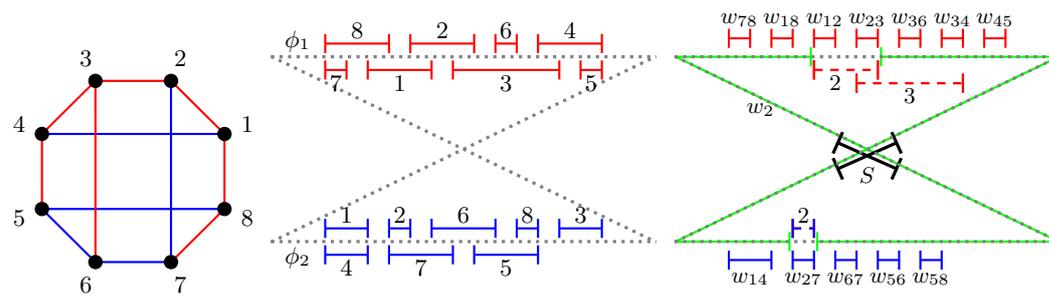
It is easy to see NP-membership [10]. To show NP-hardness, we reduce from the BIPARTITE 2-TRACK; that is to decide whether a given bipartite graph  $G$  is 2-track. A graph  $G$  is 2-track if there are sets  $E_1, E_2$  whose union is  $E(G)$  such that  $(V(G), E_1)$  and  $(V(G), E_2)$  are interval graphs. Gonçalves & Ochem proved NP-hardness of this problem [15].

*Construction:* For a given bipartite graph  $G$  we construct a graph  $G'$  that is a butterfly-graph if and only if  $G$  is 2-track. Let  $S$  be a star  $K_{1,4}$  where every edge is subdivided once. The vertex set  $V(G')$  consists of  $V(S)$ , a vertex  $w_v$  for every vertex  $v \in V(G)$ , an edge-vertex  $w_{uv}$  for every edge  $uv \in E(G)$  and  $V(S)$ . The edge set  $E(G')$  consists of  $\binom{V(G)}{2}$ ,  $E(S)$  and the edges of making  $w_{uv}$  adjacent to  $w \in V(G) \setminus \{u, v\}$  for every edge  $uv \in E(G)$ . Finally, we add every edge between  $V(G)$  and  $V(S)$ .

Given  $G$  the graph  $G'$  can be constructed in polynomial time. It remains to show that  $G$  is 2-track if and only if  $G'$  is a butterfly-graph. For the formal proof we refer to the full version of this paper. Here we only sketch how we construct a butterfly-model of  $G'$  provided  $G$  is a bipartite 2-track (witnessed by interval models  $\phi_1$  and  $\phi_2$  of  $(V(G), E_1)$  and  $(V(G), E_2)$ ). A butterfly-model of  $G'$  is obtained as follows (see Figure 5.1 for an illustration):

- we embed  $\phi_1$  and  $\phi_2$  into a subdivision  $H^\phi$  of the butterfly as shown in Figure 5.1,
- we represent the vertices of  $S$  in the center of  $H^\phi$ ,
- for  $v \in V(G)$  we represent  $w_v$  by the set  $V(H^\phi) \setminus (\phi_1(v) \cup \phi_2(v))$  (see green set  $w_2$ )
- for  $uv \in E(G)$ , we represent  $w_{uv}$  by the set  $\phi_i(u) \cap \phi_i(v)$  if  $uv \in E_i$  (see red set  $w_{23}$ ).

<sup>1</sup> Since these sets might have exponential size, the algorithm calculates only their “fingerprints”.



■ **Figure 5.1** A bipartite graph  $G$  (to the left) as a 2-track (red and blue edges) witnessed by interval models  $\phi_1$  and  $\phi_2$  (in the middle), and a *butterfly*-model of  $G'$  (to the right).

## 6 Conclusions

The question whether for a fixed graph  $H$  the class of  $H$ -graphs can be recognized in polynomial time was posed by Biro, Hujter, and Tuza over 30 years ago [5]. The main results of our work show that the boundary between polynomial and NP-hard cases of  $H$ -graphs recognition lies somewhere between unicyclic graphs  $H$ , strictly above the class of circular-arc graphs. The research carried out so far reveals connections between the  $H$ -graphs recognition problems for unicyclic graphs  $H$  and certain problems related to the Helly property of circular-arc graphs. The latter problems are now intensively studied [12] and the positive results achieved so far allow us to state the following conjecture.

► **Conjecture 6.1.** *The recognition of  $H$ -graphs is polynomial-time solvable if and only if  $H$  is a unicyclic graph or  $H$  is a tree.*

In particular, we believe that the techniques introduced in our work, suitably extended, can be used to devise polynomial algorithms for the cases where  $H$  consists of a cycle and some edges attached to it. The situation might be different when we allow to have trees attached to the cycle in  $H$ . The difficulty might be caused by the lack of a data structure maintaining all representations of a  $T$ -graph, where  $T$  is a tree different than a path (counterparts of PQ-trees for interval graphs).

---

## References

- 1 Deniz Ağaoğlu Çağırıcı and Petr Hliněný. Efficient isomorphism for  $S_d$ -graphs and  $T$ -graphs. *Algorithmica*, 2022. doi:10.1007/s00453-022-01033-8.
- 2 Deniz Ağaoğlu Çağırıcı and Petr Hliněný. Isomorphism testing for T-graphs in FPT. *WALCOM: Algorithms and Computation*, pages 239–250, 2022.
- 3 Deniz Ağaoğlu Çağırıcı and Peter Zeman. Recognition and isomorphism of proper U-graphs in FPT-time. *CoRR*, abs/2206.13372, 2022. doi:10.48550/arXiv.2206.13372.
- 4 Vikraman Arvind, Roman Nedela, Iliia Ponomarenko, and Peter Zeman. Testing isomorphism of chordal graphs of bounded leafage is fixed-parameter tractable (extended abstract). In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2022. doi:10.1007/978-3-031-15914-5\_3.
- 5 Miklós Biró, Mihály Hujter, and Zsolt Tuza. Precoloring extension. i. interval graphs. *Discret. Math.*, 100(1-3):267–279, 1992. doi:10.1016/0012-365X(92)90646-W.

- 6 K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using  $PQ$ -tree algorithms. *J. Comput. System Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
- 7 K. S. Booth.  $Pq$ -tree algorithms, November 1975. URL: <https://www.osti.gov/biblio/7189564>.
- 8 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dusan Knop, and Peter Zeman. Kernelization of graph hamiltonicity: Proper  $H$ -graphs. *SIAM J. Discret. Math.*, 35(2):840–892, 2021. doi:10.1137/19M1299001.
- 9 Steven Chaplick, Petr A. Golovach, Tim A. Hartmann, and Dusan Knop. Recognizing proper tree-graphs. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPICs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.8.
- 10 Steven Chaplick, Martin Töpfer, Jan Voborník, and Peter Zeman. On  $H$ -topological intersection graphs. *Algorithmica*, 83(11):3281–3318, 2021. doi:10.1007/s00453-021-00846-3.
- 11 Steven Chaplick and Peter Zeman. Combinatorial problems on  $h$ -graphs. *Electron. Notes Discret. Math.*, 61:223–229, 2017. doi:10.1016/j.endm.2017.06.042.
- 12 Jan Derbisz and Tomasz Krawczyk. Circular-arc graphs and the Helly property.
- 13 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on  $H$ -graphs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.30.
- 14 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. doi:10.1016/0095-8956(74)90094-X.
- 15 Daniel Gonçalves and Pascal Ochem. On star and caterpillar arboricity. *Discrete Mathematics*, 309(11):3694–3702, 2009. doi:10.1016/j.disc.2008.01.041.
- 16 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. the feedback vertex set problem. *Algorithmica*, 82(1):118–145, 2020. doi:10.1007/s00453-019-00607-3.
- 17 Haim Kaplan and Yahav Nussbaum. A simpler linear-time recognition of circular-arc graphs. *Algorithmica*, 61(3):694–737, 2011. doi:10.1007/s00453-010-9432-y.
- 18 Pavel Klavík, Jan Kratochvíl, Yota Otachi, and Toshiki Saitoh. Extending partial representations of subclasses of chordal graphs. *Theor. Comput. Sci.*, 576:85–101, 2015. doi:10.1016/j.tcs.2015.02.007.
- 19 Min Chih Lin and Jayme L. Szwarcfiter. Characterizations and linear time recognition of Helly circular-arc graphs. In *Computing and combinatorics*, volume 4112 of *Lecture Notes in Comput. Sci.*, pages 73–82. Springer, Berlin, 2006. doi:10.1007/11809678\_10.
- 20 Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. doi:10.1007/s00453-003-1032-7.
- 21 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.
- 22 Alan Tucker. An efficient test for circular-arc graphs. *SIAM J. Comput.*, 9(1):1–24, 1980. doi:10.1137/0209001.

# Descriptive Complexity for Distributed Computing with Circuits

Veeti Ahvonen   

Tampere University, Finland

Damian Heiman  

Tampere University, Finland

Lauri Hella  

Tampere University, Finland

Antti Kuusisto   

Tampere University, Finland

University of Helsinki, Finland

---

## Abstract

We consider distributed algorithms in the realistic scenario where distributed message passing is operated by circuits. We show that within this setting, modal substitution calculus MSC precisely captures the expressive power of circuits. The result is established via constructing translations that are highly efficient in relation to size. We also observe that the coloring algorithm based on Cole-Vishkin can be specified by logarithmic size programs (and thus also logarithmic size circuits) in the bounded-degree scenario.

**2012 ACM Subject Classification** Theory of computation → Finite Model Theory; Theory of computation → Distributed algorithms; Networks → Network algorithms; Theory of computation → Modal and temporal logics

**Keywords and phrases** Descriptive complexity, distributed computing, logic, graph coloring

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.9

**Related Version** *Full Version:* <https://arxiv.org/abs/2303.04735>

**Funding** Antti Kuusisto and Veeti Ahvonen were supported by the Academy of Finland project Theory of computational logics, grant numbers 352419, 352420, 353027, 324435, 328987. Antti Kuusisto was also supported by the Academy of Finland consortium project Explaining AI via Logic (XAILOG), grant number 345612. Veeti Ahvonen was also supported by the Vilho, Yrjö and Kalle Väisälä Foundation of the Finnish Academy of Science and Letters.

## 1 Introduction

Distributed computing concerns computation in networks and relates directly to various fields of study including, inter alia, cellular automata and neural networks. In this paper we study distributed systems based on circuits. A distributed system is a labeled directed graph (with self-loops allowed) where nodes communicate by sending messages to each other. In each communication round a node sends a message to its neighbours and updates its state based on (1) its own previous state and (2) the messages received from the neighbours.

Descriptive complexity of distributed computing was initiated in [8], [11] and [9]. The articles [8] and [9] characterized classes of constant-time distributed algorithms via modal logics. The constant-time assumption was lifted in [11] which showed that the expressive power of *finite message passing automata* (FMPAs) is captured by *modal substitution calculus* MSC, which is an extension of modal logic by Datalog-style rules. The papers [8], [11] and [9] did not consider *identifiers*, i.e., ID-numbers roughly analogous to IP-addresses. It is worth noting that identifiers are, for various reasons, a key concept in much of the literature on distributed computing.

 © Veeti Ahvonen, Damian Heiman, Lauri Hella, and Antti Kuusisto; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 9; pp. 9:1–9:15

 Leibniz International Proceedings in Informatics

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

In this paper we study distributed computing based on circuits in a scenario with unique identifiers. Each node runs a copy of the same circuit  $C$ . In each communication round, the node sends its current bit string  $s$  to its neighbours and updates to a new string  $s'$  by feeding  $s$  and the strings  $s_1, \dots, s_m$  sent by the neighbours to  $C$  (letting  $s'$  be the output of  $C$ ). This is a realistic model of distributed computing which also takes *local computation* – the computation steps of the circuit – into account. Typically in distributed computing, only communication steps count. Since we study distributed systems, we call our circuits *message passing circuits*, or MPCs, although formally they are just plain circuits.

We establish an exact match between this circuit-based model and the logic MSC. Unlike earlier works on descriptive complexity of distributed computing, we work in the circuit-style paradigm where an algorithm is specified via an *allocation function*  $F$  that produces, in the simplest case, for each input  $n \in \mathbb{Z}_+$ , a circuit  $F(n)$  that operates on all distributed systems (i.e., labeled directed graphs, or Kripke models) of size  $n$ . As one of our main results, we prove that programs of the MSC-logic and constant fan-in message passing circuits translate to each other with only a *linear* blow-up in size. Thus, we can work interchangeably with circuit allocation functions and MSC-program allocation functions. The related formal statements are as follows, with  $\Pi$  denoting the set of proposition symbols considered (including ones for ID-bits) while  $\Delta$  is a degree bound for graphs.

► **Theorem 12.** *Given an MPC of size  $m$  for  $(\Pi, \Delta)$ , we can construct an equivalent  $\Pi$ -program of MSC. For a constant bound  $c$  for the fan-in of MPCs, the size of the program is  $\mathcal{O}(m)$ .*

► **Theorem 13.** *Given  $\Pi$ ,  $\Delta$  and a  $\Pi$ -program of MSC of size  $m$ , we can construct an equivalent MPC for  $(\Pi, \Delta)$  of size  $\mathcal{O}(\Delta m + |\Pi|)$  when  $\Delta > 0$  and  $\mathcal{O}(m + |\Pi|)$  when  $\Delta = 0$ .*

We are especially interested in the feasible scenario where  $F(n)$  is a circuit of size  $\mathcal{O}(\log n)$ . From the above results we can prove that, for a constant  $\Delta$  and constant fan-in bound, if we have an allocation function producing log-size circuits, we also have an allocation function for log-size programs, and vice versa. We put this into use by demonstrating that for graphs of degree bound  $\Delta$ , we can produce programs of size  $\mathcal{O}(\log n)$  that compute a  $(\Delta + 1)$ -coloring via a *Cole-Vishkin* [6] style approach – implying also an analogous result for circuits.

Generally, the circuit-based approach suits well for studying the *interplay of local computation and message passing*. While important, such effects have received relatively little attention in studies on distributed computing. We provide a range of related results.

**Related work.** As already mentioned, descriptive complexity of distributed computing has been largely initiated in [9], which characterizes a range of related complexity classes via modal logics. It is shown, for example, that graded modal logic captures the class MB(1) containing problems solvable in constant time by algorithms whose recognition capacity is sufficient all the way up to distinguishing between multisets of incoming messages but no further. In the paper, the link to logic helps also in *separating* some of the studied classes. The constant-time limitation is lifted in [11], which shows that finite distributed message passing automata (FMPAs) correspond to modal substitution calculus MSC, which is the logic studied also in the current paper. The work on MSC is extended in [15], which proves that while MSC corresponds to synchronized automata, the  $\mu$ -fragment of the modal  $\mu$ -calculus similarly captures asynchronous distributed automata.

Distributed computing with identifiers has been studied from the point of view of logic earlier in [4]. The paper [4] approaches identifiers via a uniform logical characterization of a certain class of algorithms using IDs, while our work is based on the circuit-style paradigm with formulas and circuits being given based on model size. Thus the two approaches are

not comparable in any uniquely obvious way. Nevertheless, one simple difference between our work and [4] is that we treat IDs bit by bit as concrete bit strings. Thus we can express, e.g., that the current ID has a bit 1 without implying that the current node cannot have the smallest ID in the system. This is because there is no guarantee on what the set of IDs in the current graph (or distributed system) is, and in a directed graph, we cannot even scan through the graph to find out. On the other hand, the logic in [4] can express, e.g., that the current node has the largest ID, which we cannot do. Of course, with a non-uniform formula allocation function, the circuit-style paradigm can even specify non-computable properties.

The closest work to the current article is [11] which gives the already mentioned characterization of finite message passing automata via MSC. The paper does not work within the circuit-style paradigm. Furthermore, we cannot turn our circuit to an FMPA and then use the translation of [11], as this leads to an exponential blow-up in size. Also, the converse translation is non-polynomial in [11]. Furthermore, that paper does not discuss identifiers, or the Cole-Vishkin algorithm, and the work in the paper is based on the paradigm of relating properties directly with single formulae rather than our circuit-style approach. Concerning further related and *very timely* work, [2] studies graph neural networks (or GNNs) and establishes a match between *aggregate-combine* GNNs and graded modal logic. For further related work on GNNs and logic, see, e.g., [7]. Concerning yet further work on logical characterizations of distributed computing models, we mention the theses [12, 16]. For unique identifiers in graph neural networks, see [14, 17, 10].

## 2 Preliminaries

We let  $\mathbb{Z}_+$  denote the set of positive integers. For every  $n \in \mathbb{Z}_+$ , we let  $[n]$  denote the set  $\{1, \dots, n\}$  and  $[n]_0$  the set  $\{0, \dots, n\}$ . For any set  $S$ , we let  $|S|$  denote the size (or cardinality) of  $S$ . Let  $\text{PROP}$  be a countably infinite set of proposition symbols. We suppose  $\text{PROP}$  partitions into two infinite sets  $\text{PROP}_0$  and  $\text{PROP}_1$ , with the intuition that  $\text{PROP}_0$  contains ordinary proposition symbols while  $\text{PROP}_1$  consists of distinguished proposition symbols reserved for encoding ID-numbers. We denote finite sets of proposition symbols by  $\Pi \subseteq \text{PROP}$ . By  $\Pi_0$  (respectively,  $\Pi_1$ ), we mean the subset of  $\Pi$  containing ordinary (respectively, distinguished) propositions. The set  $\text{PROP}$  is associated with a linear order  $<^{\text{PROP}}$  which also induces a linear order  $<^S$  over any set  $S \subseteq \text{PROP}$ .

Let  $\Pi$  be a finite set of proposition symbols. A **Kripke model** over  $\Pi$  is a structure  $(W, R, V)$  with a non-empty domain  $W$ , an **accessibility relation**  $R \subseteq W \times W$  and a **valuation function**  $V : \Pi \rightarrow \mathcal{P}(W)$  giving each  $p \in \Pi$  a set  $V(p)$  of nodes where  $p$  is considered true. A **pointed Kripke model** is a pair  $(M, w)$  where  $M$  is a Kripke model and  $w$  a node in the domain of  $M$ . We let  $\text{succ}(w)$  denote the set  $\{v \in W \mid (w, v) \in R\}$ .

As in [9, 11], we model distributed systems by Kripke models. An edge  $(w, u) \in R$  linking the node  $w$  to  $u$  via the accessibility relation  $R$  means that  $w$  can see messages sent by  $u$ . Thereby we adopt the convention of [9, 11] that messages travel in the direction *opposite* to the edges of  $R$ . An alternative to this would be to consider modal logics with only inverse modalities, i.e., modalities based on the inverse accessibility relation  $R^{-1}$ .

We next define general notions concerning acceptance of infinite sequences of bit strings. Let  $k \in \mathbb{N}$  and consider an infinite sequence  $S = (\bar{b}_j)_{j \in \mathbb{N}}$  of  $k$ -bit strings  $\bar{b}_j$ . Let  $A \subseteq [k]$  and  $P \subseteq [k]$  be subsets, called **attention** bits and **print** bits (or bit positions, strictly speaking). Let  $(\bar{a}_j)_{j \in \mathbb{N}}$  and  $(\bar{p}_j)_{j \in \mathbb{N}}$  be the corresponding sequences of substrings of the strings in  $S$ , that is,  $(\bar{a}_j)_{j \in \mathbb{N}}$  records the substrings with positions in  $A$ , and analogously for  $(\bar{p}_j)_{j \in \mathbb{N}}$ . Let  $(\bar{r}_j)_{j \in \mathbb{N}}$  be the sequence of substrings with positions in  $A \cup P$ . We say that  $S$  **accepts** in

round  $n$  if at least one bit in  $\bar{a}_n$  is 1 and all bits in each  $\bar{a}_m$  for  $m < n$  are zero. Then also  $S$  **outputs**  $\bar{p}_n$ . More precisely,  $S$  **accepts in round  $n$  with respect to  $(k, A, P)$** , and  $\bar{p}_n$  is the **output of  $S$  with respect to  $(k, A, P)$** . The sequence  $(\bar{r}_j)_{j \in \mathbb{N}}$  is the **appointed sequence** w.r.t.  $(k, A, P)$ , and the vector  $\bar{r}_j$  the **appointed string** of round  $j$ .

We then define some logics relevant to this article. For a finite set  $\Pi$  of proposition symbols, the set of **ML( $\Pi$ )-formulas** is given by the grammar  $\varphi ::= \top \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \diamond\varphi$  where  $p \in \Pi$  and  $\top$  is a logical constant symbol. The truth of a formula  $\varphi$  in a pointed Kripke model  $(M, w)$  is defined as follows:  $(M, w) \models p \Leftrightarrow w \in V(p)$  and  $(M, w) \models \diamond\varphi \Leftrightarrow (M, v) \models \varphi$  for some  $v \in W$  such that  $(w, v) \in R$ . The semantics for  $\top, \neg, \wedge$  is the usual one.

Now, let us fix a set  $\text{VAR} := \{V_i \mid i \in \mathbb{N}\}$  of **schema variables**. We will mostly use meta variables  $X, Y, Z$ , and so on to denote symbols in  $\text{VAR}$ . The set  $\text{VAR}$  is associated with a linear order  $<^{\text{VAR}}$  inducing a corresponding linear order  $<^{\mathcal{T}}$  over any  $\mathcal{T} \subseteq \text{VAR}$ . Given a set  $\mathcal{T} \subseteq \text{VAR}$  and a set  $\Pi \subseteq \text{PROP}$ , the set of **( $\Pi, \mathcal{T}$ )-schemata** of **modal substitution calculus** (or MSC) is the set generated by the grammar  $\varphi ::= \top \mid p \mid V_i \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \diamond\varphi$ , where  $p \in \Pi$  and  $V_i \in \mathcal{T}$ . A **terminal clause** of MSC (over  $\Pi$ ) is a string of the form  $V_i(0) :- \varphi$ , where  $V_i \in \text{VAR}$  and  $\varphi \in \text{ML}(\Pi)$ . An **iteration clause** of MSC (over  $\Pi$ ) is a string of the form  $V_i :- \psi$  where  $V_i \in \text{VAR}$  and  $\psi$  is a  $(\Pi, \mathcal{T})$ -schema for some set  $\mathcal{T} \subseteq \text{VAR}$ . In a terminal clause  $V_i(0) :- \varphi$ , the symbol  $V_i$  is the **head predicate** and  $\varphi$  the **body** of the clause. Similarly,  $V_i$  is the head predicate of the iteration clause  $V_i :- \psi$  while  $\psi$  is the body.

Let  $\mathcal{T} = \{Y_1, \dots, Y_k\} \subseteq \text{VAR}$  be a finite, nonempty set of  $k$  distinct schema variables. A **( $\Pi, \mathcal{T}$ )-program**  $\Lambda$  of MSC consists of two lists

$$\begin{array}{ll} Y_1(0) :- \varphi_1 & Y_1 :- \psi_1 \\ \vdots & \vdots \\ Y_k(0) :- \varphi_k & Y_k :- \psi_k \end{array}$$

of clauses (or rules) and two sets of predicates  $\mathcal{P} \subseteq \mathcal{T}$  and  $\mathcal{A} \subseteq \mathcal{T}$ , namely **print predicates** and respectively **attention predicates** of  $\Lambda$ . The first list contains  $k$  terminal clauses over  $\Pi$  and the second contains  $k$  iteration clauses whose bodies are  $(\Pi, \mathcal{T})$ -schemata. The set  $\mathcal{P} \cup \mathcal{A}$  is the set of **appointed predicates** of  $\Lambda$ . We call  $\Lambda$  a  **$\Pi$ -program** if it is a  $(\Pi, \mathcal{T})$ -program for some  $\mathcal{T} \subseteq \text{VAR}$ . The set of head predicates of  $\Lambda$  is denoted by  $\text{HEAD}(\Lambda)$ . For each variable  $Y_i \in \text{HEAD}(\Lambda)$ , we define that  $Y_i^0 := \varphi_i$ . Recursively, assume we have defined an  $\text{ML}(\Pi)$ -formula  $Y_i^n$  for each  $Y_i \in \text{HEAD}(\Lambda)$ . The formula  $Y_i^{n+1}$  is obtained by replacing each  $Y_j$  in  $\psi_j$  by  $Y_j^n$ . Then  $Y_i^n$  is the  $n$ th **iteration formula of  $Y_i$** . More generally, if  $\varphi$  is a  $(\Pi, \mathcal{T})$ -schema, then we let  $\varphi^{n+1}$  denote the  $\text{ML}(\Pi)$ -formula obtained from the schema  $\varphi$  by simultaneously replacing each  $Y_i \in \text{HEAD}(\Lambda)$  with  $Y_i^n$ . Now, let  $(M, w)$  be a pointed  $\Pi$ -model. We define that  $(M, w) \models \Lambda$  if for some  $n$  and some attention predicate  $Y$  of  $\Lambda$ , we have  $(M, w) \models Y^n$ . In Section 3, we will also define output conditions for MSC using print predicates.

For every  $(\Pi, \mathcal{T})$ -schema  $\psi$ , we let  $\text{md}(\psi)$  denote the **modal depth** of  $\psi$  (i.e., the maximum nesting depth of diamonds  $\diamond$  in  $\psi$ ). We let  $\text{mdt}(\Lambda)$  (respectively,  $\text{mdi}(\Lambda)$ ) denote the maximum modal depth of the bodies of the terminal clauses (resp., of the iteration clauses) of  $\Lambda$ . By  $\text{SUBS}(\Lambda)$  we denote the set of all subschemata of  $\Lambda$ , including head predicates and bodies of iteration and terminal clauses. If  $S$  is a set of schemata,  $\text{SUBS}(S)$  is the set of all subschemata of all schemata in  $S$ .

► **Example 1.** Given a proposition symbol  $p$  and a pointed Kripke model  $(M, w)$ , we say that  $p$  is *reachable* from  $w$  if there exists a directed path from  $w$  to a node  $v$  in  $M$  such that  $(M, v) \models p$ . Now, consider the program  $X(0) :- p, \quad X :- \diamond X$  where  $X$  is the appointed predicate. It is easy to show that  $(M, w) \models X^j$  for some  $j < n$  if and only if  $p$  is reachable from  $w$ , where  $n$  is the domain size of  $M$ .

Next we define a class of Kripke models which includes identifiers that are encoded by proposition symbols. Assume that  $p_1, \dots, p_\ell$  enumerate all the distinguished propositions in  $\Pi$  in the order  $<^{\text{PROP}}$ . For each node  $w$  of a Kripke model  $M$  over  $\Pi$ , we let  $\text{ID}(w)$  denote the **identifier** of  $w$ , that is, the  $|\Pi_1|$ -bit string such that the  $i$ th bit of  $\text{ID}(w)$  is 1 if and only if  $(M, w) \models p_i$ . The model  $M$  is a Kripke model **with identifiers** if  $\text{ID}(w) \neq \text{ID}(w')$  for each pair of distinct nodes  $w$  and  $w'$  of  $M$ . We let  $\mathcal{K}(\Pi, \Delta)$  denote the class of finite Kripke models  $(W, R, V)$  over  $\Pi$  with identifiers such that the out-degree of each node is at most  $\Delta \in \mathbb{N}$ . For a node  $w$ , let  $s_1, \dots, s_d$  be the identifiers of the members of  $\text{succ}(w)$  in the lexicographic order. A node  $v \in \text{succ}(w)$  is the  **$i$ th neighbour** of  $w$  iff  $\text{ID}(v) = s_i$ . Analogously to  $\text{ID}(w)$ , if  $p_1, \dots, p_m$  enumerate all the propositions in  $\Pi$  in the order  $<^{\text{PROP}}$ , then the **local input** of a node  $w$  of a Kripke model  $M$  over  $\Pi$  is the  $m$ -bit string  $t$  such that the  $i$ th bit of  $t$  is 1 if and only if  $(M, w) \models p_i$ .

## 2.1 Circuits and distributed computation

Here we first recall some basics related to circuits and then define a related distributed computation model. A **Boolean circuit** is a directed acyclic graph where each node of non-zero in-degree is labeled by one of the symbols  $\wedge, \vee, \neg$ . The nodes of a circuit are called **gates**. The in-degree of a gate  $u$  is called the **fan-in** of  $u$ , and the out-degree of  $u$  is **fan-out**. The **input gates** of a circuit are precisely the gates that have zero fan-in; these gates are not labeled by  $\wedge, \vee, \neg$ . The **output-gates** are the ones with fan-out zero; we allow multiple output gates in a circuit. Note that gates with  $\wedge, \vee$  can have any positive fan-in (also 1). The fan-in of every gate labeled with  $\neg$  is 1. The **size**  $|C|$  of a circuit  $C$  is the number of gates in  $C$ . The **depth**  $d(C)$  of  $C$  is the longest path length (number of edges) from an input gate to an output gate. The **height**  $h(G)$  of a gate  $G$  in  $C$  is the longest path length from an input gate to the gate  $G$ . Thus the height of an input gate is zero. Both the input gates and output gates of a circuit are linearly ordered. A circuit with  $n$  input gates and  $k$  output gates then computes a function of type  $\{0, 1\}^n \rightarrow \{0, 1\}^k$ . This is done in the natural way, analogously to the Boolean operators corresponding to  $\wedge, \vee, \neg$ , see for example [13] for the formal definition. The output of the circuit is the *binary string* determined by the output bits of the output gates.

From a Boolean formula it is easy to define a corresponding circuit by considering its *inverse tree representation*, meaning the tree representation with edges pointing in the inverse direction (toward the root). A node  $v$  in the inverse tree representation is the **parent** of  $w$  if there is an edge from  $w$  to  $v$ . Then  $w$  is a **child** of  $v$ . Note that input gates do not have any children and output gates have no parents. The **descendants** of  $w$  are defined such that every child of  $w$  is a descendant of  $w$  and also every child of a descendant of  $w$  is a descendant of  $w$ .

► **Definition 2.** Let  $\Pi$  be a set of propositions and  $\Delta \in \mathbb{N}$ . A **circuit for  $(\Pi, \Delta)$**  is a circuit  $C$  that specifies a function  $f : \{0, 1\}^{|\Pi|+k(\Delta+1)} \rightarrow \{0, 1\}^k$  for some  $k \in \mathbb{N}$ . The number  $k$  is called the **state length** of  $C$ . The circuit  $C$  is also associated with sets  $A \subseteq [k]$  and  $P \subseteq [k]$  of **attention bits** and **print bits**, respectively. For convenience, we may also call a circuit  $C$  for  $(\Pi, \Delta)$  a **message passing circuit** (or MPC) for  $(\Pi, \Delta)$ . The set  $A \cup P$  is called the **set of appointed bits** of the circuit.

A circuit  $C$  is **suitable** for a Kripke model  $M \in \mathcal{K}(\Pi, \Delta')$  with identifiers if  $C$  is a message passing circuit for  $(\Pi, \Delta)$  for some  $\Delta \geq \Delta'$ . A circuit  $C$  for  $(\Pi, \Delta)$  with  $|\Pi_1| = m$  is referred to as a **circuit for  $m$  ID-bits**. We let  $\text{CIRC}(\Pi_0, \Delta)$  denote the set of all circuits  $C$  such

that, for some  $\Pi$  with  $\Pi \cap \text{PROP}_0 = \Pi_0$ , the circuit  $C$  is a circuit for  $(\Pi, \Delta)$ . We stress that strictly speaking, when specifying an MPC, we should always specify (together with a circuit) the sets  $\Pi$ ,  $\Delta$ , the attention and print bits, and an ordering of the input and output gates.

Before giving a formal definition of distributed computation in a Kripke model  $M \in \mathcal{K}(\Pi, \Delta)$  with a circuit  $C$  for  $(\Pi, \Delta)$ , we describe the process informally. Each node  $u$  of  $M$  runs a copy of the circuit  $C$ . The node  $u$  is associated with a local input, which is the binary string that corresponds to the set of propositions true at  $u$ . In the beginning of computation, the circuit at  $u$  reads the string  $\bar{s} \cdot 0^\ell$  at  $u$ , where  $\bar{s}$  is the local input at  $u$  and  $\ell = k(\Delta + 1)$ , so  $0^\ell$  is simply the part of the input to  $C$  that does not correspond to proposition symbols. Then the circuit enters a *state* which is the  $k$ -bit output string of  $C$ . Let  $s(0, u)$  denote this string and call it the **state in communication round 0** at the node  $u$ . Now, recursively, suppose we know the state  $s(n, u)$  in communication round  $n \in \mathbb{N}$  for each node  $u$ . The state  $s(n+1, u)$  for round  $n+1$  at  $u$  is then computed as follows.

1. At each node  $u$ , the circuit sends  $s(n, u)$  to the nodes  $w$  such that  $R(w, u)$ . Note here that messages flow opposite to the direction of  $R$ -edges.
2. The circuit at  $u$  *updates* its state to  $s(n+1, u)$  which is the  $k$ -bit string obtained as the output of the circuit with the input  $\bar{s} \cdot \bar{s}_0 \cdots \bar{s}_\Delta$  which is the concatenation of the  $k$ -bit strings  $s_i$  (for  $i \in \{0, \dots, \Delta\}$ ) specified as follows. The string  $\bar{s}$  is the local input at  $u$ . The string  $\bar{s}_0$  is the state  $s(n, u)$ . Let  $i \in \{1, \dots, m\}$ , where  $m \leq \Delta$  is the out-degree of  $u$ . Then  $\bar{s}_i$  is the state  $s(n, v_i)$  of the  $i$ th neighbour  $v_i$  of  $u$ . For  $i > m$ , we have  $\bar{s}_i = 0^k$ .

We then define computation of MPCs formally. An MPC  $C$  for  $(\Pi, \Delta)$  of state length  $k$  and a Kripke model  $M = (W, R, V) \in \mathcal{K}(\Pi, \Delta)$  define a *synchronized distributed system* which executes an  $\omega$ -sequence of rounds defined as follows. Each round  $n \in \mathbb{N}$  defines a **global configuration**  $f_n: W \rightarrow \{0, 1\}^k$ . Let  $\bar{t}_w$  denote the binary string corresponding to the set of propositions true at  $w$  (i.e., local input). The configuration of round 0 is the function  $f_0$  such that  $f_0(w)$  is the  $k$ -bit binary string produced by  $C$  with the input  $\bar{t}_w \cdot 0^{k(\Delta+1)}$ . Recursively, assume we have defined  $f_n$ . Let  $v_1, \dots, v_m \in \text{succ}(w)$  be the neighbours of  $w$  ( $m \leq \Delta$ ) given in the order of their IDs. Let  $\bar{s}_w$  be the concatenation  $\bar{t}_w \cdot \bar{s}_0 \cdots \bar{s}_\Delta$  of  $k$ -bit binary strings such that **(1)**  $\bar{s}_0 = f_n(w)$ , **(2)**  $\bar{s}_i = f_n(v_i)$  for each  $i \in \{1, \dots, m\}$ , **(3)**  $\bar{s}_j = 0^k$  for  $j \in \{m+1, \dots, \Delta\}$ . Then  $f_{n+1}(w)$  is the output string of  $C$  with input  $\bar{s}_w$ . Now, consider the sequence  $(f_n(w))_{n \in \mathbb{N}}$  of  $k$ -bit strings that  $C$  produces at  $w$ . Suppose the sequence  $(f_n(w))_{n \in \mathbb{N}}$  accepts (resp. outputs  $\bar{p}$ ) in round  $n$  w.r.t.  $(k, A, P)$ . Then  $w$  **accepts** (resp., **outputs**  $\bar{p}$ ) in round  $n$ . Note that the circuit at  $w$  keeps executing after round  $n$ .

Given a Kripke model  $M = (W, R, V)$ , a **solution labeling** is a function  $W \rightarrow \{0, 1\}^*$  associating nodes with strings. The strings represent outputs of the nodes on distributed computation. We could, e.g., label the nodes with strings corresponding to “yes” and “no”. A **partial solution labeling** for  $M$  is a partial function from  $W$  to  $\{0, 1\}^*$ , that is, a function of type  $U \rightarrow \{0, 1\}^*$  for some  $U \subseteq W$ . Partial solution labelings allow for “divergent computations” on some nodes in  $W$ . The **global output** of a circuit  $C$  over a model  $M = (W, R, V)$  is a function  $g: U \rightarrow \{0, 1\}^*$  such that **(1)**  $U \subseteq W$ , **(2)** for all  $w \in U$ , the circuit  $C$  outputs  $g(w)$  in some round  $n$ , and **(3)**  $C$  does not produce an output for any  $v \in W \setminus U$ . Now, fix a finite set  $\Pi_0 \subseteq \text{PROP}_0$  of proposition symbols. Intuitively, these are the “actual” propositions in models, while the set of ID-propositions will grow with model size. Let  $\mathcal{M}(\Pi_0)$  denote the class of all finite Kripke models  $M$  with IDs and having a set  $\Pi$  of proposition symbols such that  $\Pi \cap \text{PROP}_0 = \Pi_0$ . Thus  $\Pi_0$  is the same for all models in  $\mathcal{M}(\Pi_0)$  but the symbols for IDs vary. Consider a subclass  $\mathcal{M} \subseteq \mathcal{M}(\Pi_0)$ . Now, a distributed computing **problem** over  $\mathcal{M}$  is a mapping  $p$  with domain  $\mathcal{M}$  that associates to each input  $M$  a (possibly infinite) set  $p(M)$  of partial solution labelings for  $M$ . The set  $p(M)$  represents the set of acceptable answers to the problem  $p$  over  $M$ . Many graph problems (e.g., colorings) naturally involve a set of such answer labelings.

For  $\Delta \in \mathbb{N}$ , we let  $\mathcal{M}(\Pi_0, \Delta)$  denote the restriction of  $\mathcal{M}(\Pi_0)$  to models with maximum out-degree  $\Delta$ . A **circuit sequence** for  $\mathcal{M}(\Pi_0, \Delta)$  is a function  $F : \mathbb{Z}_+ \rightarrow \text{CIRC}(\Pi_0, \Delta)$  such that  $F(n)$  is a circuit for  $\lceil \log n \rceil$  ID-bits. Now,  $F$  **solves** a problem  $p$  over  $\mathcal{M}(\Pi_0, \Delta)$  if the global output of  $F(n)$  belongs to  $p(M)$  for each  $M \in \mathcal{M}(\Pi_0, \Delta)$  of domain size  $n$ . Let  $c \in \mathbb{N}$ . We define  $\text{DCC}_\Delta^c[\log n]$  to be the class of distributed computing problems solvable by a circuit sequence  $F$  for some  $M \in \mathcal{M}(\Pi_0, \Delta)$  of maximum fan-in  $c$  circuits such that the size of  $F(n)$  is  $\mathcal{O}(\log n)$ . The related LOGSPACE uniform class requires that each  $F$  can be computed in LOGSPACE. DCC stands for *distributed computing by circuits*. Note that circuit sequences for  $\text{DCC}_\Delta^c[\log n]$  are trivially sequences for  $\text{NC}^1$ .

### 3 Extensions of MSC

The rest of this article is basically a proof of the expressive equivalence of MSC and MPCs over distributed systems, with a small blow-up in the respective sizes of programs and circuits. The argument is long, but we have divided it into suitably short lemmas to improve readability. The argument splits into the following two main parts:

1. equivalence of MPCs and **message passing MSC**, or MPMSC, an auxiliary logic to be defined below,
2. equivalence of MPMSC and MSC.

MPMSC is mainly used as a tool, and indeed, MPMSC and the related notions greatly help shorten and organize our arguments.

We define MPMSC via two further auxiliary logics. Let  $\Pi$  be a set of propositions and  $\mathcal{T}$  a set of schema variables. Let  $\Delta \in \mathbb{N}$ . In **Multimodal MSC** (or MMSC), instead of  $\diamond$ , we have the operators  $\diamond_1, \dots, \diamond_\Delta$ , and otherwise the syntax is as in MSC. The schema  $\diamond_i \varphi$  simply asks if  $\varphi$  is true at the  $i$ th neighbour. More formally, if  $(M, w)$  is a pointed Kripke model with identifiers, then  $(M, w) \models \diamond_i \varphi \Leftrightarrow (M, v_i) \models \varphi$  such that  $(w, v_i) \in R$  and  $v_i$  is the  $i$ th neighbour of  $w$ , noting that if the out-degree of  $w$  is less than  $i$ , then  $\diamond_i \varphi$  is false at  $w$ . A  **$(\Pi, \Delta)$ -program** of MMSC is exactly like a  $\Pi$ -program of MSC but we are only allowed to use operators  $\diamond_1, \dots, \diamond_\Delta$  instead of  $\diamond$ . A  **$\Pi$ -program**  $\Lambda$  of MMSC is a  $(\Pi, \Delta)$ -program for any  $\Delta \geq d$ , where  $d$  is the maximum subindex in any diamond in  $\Lambda$ . We also fix print and attention predicates for programs of MMSC. Note that MMSC is not a logic in the usual sense as the operators  $\diamond_i$  require information about the predicates defining IDs. This could be remedied via signature changes and limiting attention to multimodal models with relations having out-degree at most one. This approach would be a bit messy, and the current approach suffices for this article.

We next define MSC **with conditional rules** (or CMSC). Here we allow “if-else” rules as iteration clauses. Let  $\varphi_1, \dots, \varphi_n$  and  $\psi_1, \dots, \psi_n$  and also  $\chi$  be  $(\Pi, \mathcal{T})$ -schemata of basic MSC. A **conditional iteration clause** is a rule of the form  $X :=_{\varphi_1, \dots, \varphi_n} \psi_1; \dots; \psi_n; \chi$ . The schemata  $\varphi_i$  are **conditions** for the head predicate  $X$  and the schemata  $\psi_i$  are the related **consequences**. The last schema  $\chi$  is called the **backup**. Note that when  $n = 0$ , we have a standard MSC clause.  **$\Pi$ -programs** of CMSC are exactly as for MSC, but we are allowed to use conditional iteration clauses. Thus a program  $\Lambda$  of CMSC consists of  $k$  terminal clauses,  $k' \leq k$  conditional iteration clauses and  $k - k'$  standard iteration clauses for some  $k \in \mathbb{Z}_+$ . Again we also fix some sets of schema variables as print and attention predicates.

To fix the semantics, we will specify – as in MSC – the  $n$ th iteration formula of each head predicate. Informally, we always use the first (from the left) condition  $\varphi_i$  that holds and thus evaluate the corresponding consequence  $\psi_i$  as the body of our rule. If none of the conditions hold, then we use the backup. Let  $\Lambda$  be a  $\Pi$ -program of CMSC. First, we let

the zeroth iteration clause  $Y_i^0$  of a head predicate  $Y_i \in \text{HEAD}(\Lambda)$  be the terminal clause of  $Y_i$ . Recursively, assume we have defined an ML( $\Pi$ )-formula  $Y_i^n$  for each  $Y_i \in \text{HEAD}(\Lambda)$ . Now, consider the rule  $Y_i : -\varphi_1, \dots, \varphi_m \ \psi_1; \dots; \psi_m; \chi$ . Let  $\varphi_j^{n+1}$  be the formula obtained by replacing each schema variable  $Y_k$  in the condition  $\varphi_j$  by  $Y_k^n$ . The formulae  $\chi^{n+1}$  and  $\psi_k^{n+1}$  are obtained analogously. Then, the formula  $Y_i^{n+1}$  is

$$\bigvee_{k \leq m} \left( \left( \bigwedge_{j < k} \neg \varphi_j^{n+1} \right) \wedge \varphi_k^{n+1} \wedge \psi_k^{n+1} \right) \vee \left( \left( \bigwedge_{j \leq m} \neg \varphi_j^{n+1} \right) \wedge \chi^{n+1} \right).$$

Often the backup schema  $\chi$  is just the head predicate  $X$  of the rule. This means the truth value of the head predicate does not change if none of the conditions hold. We say that a condition  $\varphi_k$  is **hot** at  $w$  in round  $n \geq 1$  if the formula  $\varphi_k^n$  is true at  $w$  and none of the “earlier” formulas  $\varphi_j^n$  for conditions of the same rule (so  $j < k$ ) are true. Otherwise the backup is hot. We call a conditional iteration clause (or the corresponding head predicate) **active** in round  $n \geq 1$  at node  $w$  if one of the condition formulas of the rule is hot.

We finally specify **message passing** MSC (or MPMSC) essentially as multimodal MSC with conditional rules. The  $(\Pi, \Delta)$ -**programs** are exactly like  $(\Pi, \Delta)$ -programs of MMSC with conditional rules and the following restrictions. **(1)** The modal depth of terminal clauses and conditions of rules is zero. **(2)** The consequences, backups and bodies of standard iteration clauses all have modal depth at most one. As in MMSC, operators  $\diamond$  are not allowed. A  $\Pi$ -program of MPMSC is defined analogously to a  $\Pi$ -program MMSC. Thus a program of MPMSC contains  $k$  terminal clauses,  $k' \leq k$  conditional iteration clauses and  $k - k'$  standard iteration clauses for some  $k \in \mathbb{Z}_+$ . We also fix sets of attention and print predicates. The semantics is defined as for CMSC, noting that now diamonds  $\diamond_i$  are used. A non-terminal clause of a program of MPMSC is a **communication clause** if it contains at least one diamond. A communication clause is **listening** in round  $n \in \mathbb{Z}_+$  if one of the following holds. **(1)** A condition  $\varphi_i$  is hot and the corresponding consequence has a diamond. **(2)** A backup is hot and has a diamond. **(3)** The rule is not conditional but has a diamond.

### 3.1 Notions of equivalence and acceptance

Here we introduce useful acceptance and output conditions for programs of all variants of MSC, including standard MSC. The acceptance conditions will be consistent with the already given conditions for standard MSC.

Let  $\Lambda$  be a program and  $\mathcal{A}$  and  $\mathcal{P}$  the sets of attention and print predicates. Let  $Y_1, \dots, Y_k$  enumerate the head predicates in  $\Lambda$  in the order  $<^{\text{VAR}}$ . Let  $M = (W, R, V)$  be a Kripke model. Each round  $n \in \mathbb{N}$  defines a **global configuration**  $g_n : W \rightarrow \{0, 1\}^k$  given as follows. The configuration of the  $n$ th round is the function  $g_n$  such that the  $i$ th bit of  $g_n(w)$  is 1 if and only if  $(M, w) \models Y_i^n$ . If the sequence  $(g_n(w))_{n \in \mathbb{N}}$  accepts (respectively outputs  $\bar{p}$ ) in round  $n$  with respect to  $(k, \mathcal{A}, \mathcal{P})$ , then we say that the node  $w$  **accepts** (respectively **outputs**  $\bar{p}$ ) in round  $n$ . Then  $n$  is the **output round** (also called the **computation time**) of  $\Lambda$  at  $w$ . Note that the output round is a unique round since the accepting round is unique by the definition of infinite bit sequences where print and attention bits are fixed. We write  $(M, w) \models \Lambda$  if node  $w$  accepts in some round  $n$ . For a program  $\Lambda$  of message passing MSC and model  $M$ , a **global communication round** is a computation round  $n$  where at least one communication clause is listening in at least one node of  $M$ . A program  $\Lambda$  **outputs**  $\bar{p}$  **at**  $w$  **in global communication time**  $m$  if the output round of  $\Lambda$  at  $w$  is  $n$  and  $m \leq n$  is the number of global communication rounds in the set  $\{0, \dots, n\}$  of rounds in the computation.

Now, let  $\mathcal{L}$  denote the set of all programs of all of our variants of MSC. Let  $\mathcal{C}$  denote the set of all MPCs. For each  $\Lambda \in \mathcal{L}$ , we say that a Kripke model  $M$  is **suitable** for  $\Lambda$  if  $M$  interprets (at least) all the proposition symbols that occur in  $\Lambda$ . For a message passing

circuit for  $(\Pi, \Delta)$ , we say that  $M$  is **suitable** for the circuit if the set of proposition symbols interpreted by  $M$  is precisely  $\Pi$  and the maximum out-degree of  $M$  is at most  $\Delta$ . Now, let  $x$  and  $y$  be any members of  $\mathcal{C} \cup \mathcal{L}$ . We say that  $x$  and  $y$  are (acceptance) **equivalent** if for each Kripke model  $M$  that is suitable for both  $x$  and  $y$  and for each node  $w$  in the model,  $x$  and  $y$  produce the same output at  $w$  or neither produce any output at all at  $w$ . We say that  $x$  and  $y$  are **strongly equivalent**, if for each  $M$  suitable for  $x$  and  $y$  and for each node  $w$  in the model and in every round  $n$ , the objects  $x$  and  $y$  produce the same appointed string  $\bar{r}_n$  at  $w$ . We also define a special *weakened equivalence* notion for MPMSC and MPC. We say that a program  $\Lambda$  of MPMSC and a circuit  $C$  are **strongly communication equivalent**, if for each  $M$  suitable for both  $\Lambda$  and  $C$  and for each node  $w$  in the model, the appointed sequence  $S$  of the circuit is precisely the sequence  $(\bar{r}_j)_{j \in G}$  of appointed strings of the program, where  $G \subseteq \mathbb{Z}_+$  is the set of global communication rounds  $n$  of the program. Moreover, the MPMSC must not accept in any non-communication round. Finally, the **length** or **size** of a program (respectively, a schema) of any variant of MSC is the number of *occurrences* of proposition symbols, head predicates, and operators  $\top$ ,  $\neg$ ,  $\wedge$ ,  $\diamond$ ,  $\diamond_i$ . The modal depth  $\text{md}(\Lambda)$  of a program  $\Lambda$  is the maximum modal depth of its rule bodies (iteration and terminal).

## 4 Linking MPMSC to message passing circuits

To obtain the desired descriptive characterizations, we begin by translating MPCs to MPMSC.

### 4.1 From MPC to MPMSC

To ultimately translate MPCs to MPMSC, we will first show how to simulate the evaluation of a standard Boolean circuit with a diamond-free program of MSC. Let  $C$  be a circuit of depth  $d$  with  $\ell$  input and  $k$  output gates. Let  $L$  denote any of the variants of MSC. Fix schema variables  $I_1, \dots, I_\ell$  and  $O_1, \dots, O_k$ , with both sequences given here in the order  $<^{\text{VAR}}$ . Consider a program  $\Lambda$  of  $L$  with the following properties.

1. The set of schema variables of  $\Lambda$  contains (at least) the variables  $I_1, \dots, I_\ell, O_1, \dots, O_k$ .
2. The program has no diamond operators ( $\diamond$  or  $\diamond_i$ ) and contains no proposition symbols.
3. The terminal clause for each schema variable  $X$  is  $X(0) :- \perp$ .

Let  $P: \{\perp, \top\}^\ell \rightarrow \{\perp, \top\}^k$  be the function defined as follows. For each input  $(x_1, \dots, x_\ell) \in \{\perp, \top\}^\ell$  to  $P$ , modify  $\Lambda$  to a new program  $\Lambda(x_1, \dots, x_\ell)$  by changing each terminal clause  $I_i(0) :- \perp$  to  $I_i(0) :- x_i$ . Let  $(y_1, \dots, y_k) \in \{\perp, \top\}^k$  be the tuple of truth values of the  $d$ th iteration formulas  $O_1^d, \dots, O_k^d$ , where we recall that  $d$  is the depth of our circuit  $C$ . Then we define  $P(x_1, \dots, x_\ell) := (y_1, \dots, y_k)$ . Now, if  $P$  defined this way is identical to the function computed by  $C$ , then  $\Lambda$  **simulates** the circuit  $C$  (w.r.t.  $I_1, \dots, I_\ell$  and  $O_1, \dots, O_k$ ).

► **Lemma 3.** *For each circuit  $C$  of size  $m$  and with  $n$  edges, there exists a program of  $L$  of size  $\mathcal{O}(m+n)$  that simulates  $C$ , where  $L$  is any of the variants of MSC. Furthermore, with constant fan-in, the size of the program is  $\mathcal{O}(m)$ .*

**Proof.** Assume first that the depth  $d$  of  $C$  is at least 1. Next we modify  $C$  so that we obtain a circuit  $C'$  with the following properties: **(1)** The height of each output gate is the same, **(2)** the depth of  $C'$  is  $\mathcal{O}(d)$ , **(3)** the size of  $C'$  is  $\mathcal{O}(|C|)$  and **(4)**  $C'$  specifies the same function as  $C$ . The formal construction of  $C'$  is given in [1]. Then we define a schema variable for each gate of  $C'$ . The variables for the input gates are  $I_1, \dots, I_\ell$  while those for the output gates are  $O_1, \dots, O_k$ . Let  $X$  be a schema variable for a  $\wedge$ -gate  $G$  of  $C'$ . We define a corresponding terminal clause  $X(0) :- \perp$  and iteration clause  $X :- Y_1 \wedge \dots \wedge Y_j$ , where  $Y_1, \dots, Y_j$  are the variables for the gates that connect to  $G$ . With constant fan-in we have a constant amount of

connecting gates and therefore the length of each rule is  $\mathcal{O}(1)$ . Similarly, for a variable  $X'$  for a disjunction gate  $G'$ , we define the rules  $X'(0) :- \perp$  and  $X' :- Y'_1 \vee \dots \vee Y'_j$  where  $Y'_1, \dots, Y'_j$  are the variables for the gates connecting to  $G'$ . For negation, we define  $X''(0) :- \perp$  and  $X'' :- \neg Y$ , where  $Y$  is the variable for the connecting gate. We let the terminal clauses for the head predicates  $I_i$  relating to input gates be  $I_i(0) :- \perp$ . This choice of rules is irrelevant, as when checking if a program simulates a circuit, we modify the terminal rules to match input strings. The related iteration clause is  $I_i :- I_i$ .

Finally, in the extreme case where the depth of  $C$  is 0 (each input gate is also an output gate), we define the program with the head predicate sequence  $(I_1, \dots, I_\ell) = (O_1, \dots, O_k)$  and such that the (terminal and iteration) clause for each head predicate  $I_i = O_i$  is  $I_i :- \perp$ . ◀

► **Theorem 4.** *Given an MPC for  $(\Pi, \Delta)$  of size  $m$ , we can construct a strongly communication equivalent  $(\Pi, \Delta)$ -program of MPMSC. Supposing a constant bound  $c$  for the fan-in of MPCs, the size of the program is linear in the size of the circuit. Moreover, the computation time is  $\mathcal{O}(d)$  times the computation time of the MPC, where  $d$  is the depth of the MPC.*

**Proof.** Let  $C$  be an MPC for  $(\Pi, \Delta)$  of state length  $k$ . We will first explain informally how our program  $\Lambda_C$  for the circuit  $C$  will work. The program  $\Lambda_C$  uses  $k$  head predicates to simulate the state of the circuit. We will use Lemma 3 to build our program, and the operators  $\diamond_i$  will be used to simulate receiving messages of neighbours. The program  $\Lambda_C$  computes in repeated periods of  $d + 1$  rounds, where  $d = d(C)$  is the depth of  $C$ . Simulating the reception of neighbours' messages takes one round, and the remaining  $d$  rounds go to simulating the evaluation of the circuit.

Now we define our program formally. First we define a clock; the idea is for  $\Lambda_C$  to simulate the computation of  $C$  once per each cycle of the clock. We assume that the depth of  $C$  is at least 1, because if it is 0 then the clock is omitted and the rules of the program are trivial to construct. The clock consists of the head predicates  $T_0, T_1, \dots, T_{d(C)}$  and the following rules:  $T_0(0) :- \perp$ ,  $T_0 :- T_{d(C)}$ ,  $T_1(0) :- \top$ ,  $T_1 :- T_0$  and for  $i \in [d(C) - 1]$ , we have  $T_{i+1}(0) :- \perp$  and  $T_{i+1} :- T_i$ . In every round, precisely one of the head predicates  $T_i$  is true and the others are false. In round 0, the only true predicate is  $T_1$ , and in round  $i \in [d(C) - 1]$ , the only true predicate is  $T_{i+1}$ . After  $d(C)$  rounds the predicate  $T_0$  is true, and in the next round the clock starts over again.

Let  $\Gamma_C$  be a program simulating the internal evaluation of the circuit  $C$  as given in the proof of Lemma 3. We will obtain  $\Lambda_C$  by using the clock and rewriting some of the iteration clauses of  $\Gamma_C$  as follows. If  $X_G$  is a head predicate corresponding to a non-input gate  $G$  in  $\Gamma_C$ , then we rewrite the corresponding iteration clause  $X_G :- \varphi$  to  $X_G :-_{T_{h(G)}} \varphi; X_G$ , where  $h(G)$  is the height of the gate  $G$ .

For every  $\ell \in [|\Pi|]$ , we let  $I_\ell^\Pi$  refer to the head predicate of  $\Gamma_C$  that corresponds to the input gate of  $C$  that reads the truth value of proposition  $p_\ell$ . For every  $i \in [k]$  and  $j \in [\Delta]_0$  we let  $I_{(i,j)}$  refer to a head predicate of  $\Gamma_C$  that corresponds to the input gate of  $C$  that reads the  $i$ th value of the state string of the  $j$ th neighbour. The “neighbour 0” refers to the home node. Next, we will rewrite the clauses with head predicates corresponding to input gates. For every  $i \in [k]$ , we let  $O_i$  refer to the head predicate of  $\Gamma_C$  that corresponds to the  $i$ th output gate of  $C$ . The terminal (respectively, iteration) clause for  $I_i^\Pi$  is rewritten to be  $I_i^\Pi(0) :- p_i$  (resp.,  $I_i^\Pi :-_{T_0} p_i; I_i^\Pi$ ). If  $j \neq 0$ , then the terminal (resp., iteration) clause for every  $I_{(i,j)}$  is rewritten to be  $I_{(i,j)}(0) :- \perp$  (resp.,  $I_{(i,j)} :-_{T_0} \diamond_j O_i; I_{(i,j)}$ ). The terminal (resp., iteration) clause for every  $I_{(i,0)}$  is rewritten to be  $I_{(i,0)}(0) :- \perp$  (resp.,  $I_{(i,0)} :-_{T_0} O_i; I_{(i,0)}$ ). Now, we have obtained the iteration and terminal clauses of  $\Lambda_C$ .

The attention and print predicates of  $\Lambda_C$  are defined as follows. Let  $A \subseteq [k]$  (resp.  $P \subseteq [k]$ ) be the set of the attention (resp., print) bit positions in  $C$ . The print predicates of  $\Lambda_C$  are precisely the head predicates  $O_j$ , where  $j \in P$ . If the depth of  $C$  is 0, then the attention predicates of  $\Lambda_C$  are precisely the head predicates  $O_j$ , where  $j \in A$ . If the depth of  $C$  is greater than 0, then we add a fresh attention predicate  $A'$  whose terminal clause is  $A'(0) :- \perp$  and whose iteration clause is the disjunction of the head predicates  $O_j$  where  $j \in A$ . This is done to ensure that our program accepts during a communication round.

We analyze how  $\Lambda_C$  works. The program executes in a periodic fashion in cycles with  $d(C) + 1$  rounds in each cycle. In round 0, the program  $\Lambda_C$  reads the proposition symbols and records the local input with the head predicates  $I_i^{\Pi}$  whose truth values will remain constant for the rest of the computation. Also,  $T_1$  evaluates to true in round 0. In round 1, the head predicates corresponding to gates at height one are active and thus updated. (Note that the predicates  $I_{(i,j)}$  for input gates are inactive because  $T_0$  is false, so they stay false in round 1, because in round 0 they evaluate to false and the backup has no effect on the truth value.) From height one, the execution then continues to predicates for gates at height two, and so on. In round  $d(C)$ , the head predicates for output gates  $O_i$  are active. The program also outputs if an attention predicate is true. In round  $d(C) + 1$ , the predicate  $T_0$  is true and thus the input gate predicates  $I_{(i,j)}$  are active, and thereby the program starts again by updating them using diamonds  $\diamond_i$ . They obtain truth values that correspond to an input string to our circuit. The program then proceeds to simulate height one in round  $d(C) + 2$ , continuing in further rounds all the way up to height  $d(C)$  gates and finishing the second cycle of the execution of  $\Lambda_C$ . The subsequent cycles are analogous. Thus our program  $\Lambda_C$  simulates  $C$  in a periodic fashion.

It is easy to check that the program  $\Lambda_C$  is strongly communication equivalent to  $C$ . The communication clauses in  $\Lambda_C$  are synchronous, i.e., all nodes are listening in the same rounds. This is because simulating the circuit takes the same amount of time at every node. The translation is clearly linear in the size of  $C$  (for constant fan-in  $C$ ) due to Lemma 3. ◀

## 4.2 From MPMSC to MPC

Converting an MPMSC-program to a circuit is, perhaps, easier. The state string of the constructed MPC essentially stores the values of the head predicates and proposition symbols used by the program and computes a new state string by simulating the program clauses. We begin with the following lemma that shows how to get rid of conditional rules.

► **Lemma 5.** *Given a  $\Pi$ -program of CMSC, we can construct a strongly equivalent  $\Pi$ -program of MSC of size linear in the size of the CMSC-program and with the same maximum modal depth in relation to both terminal and iteration clauses.*

**Proof.** The full proof – given in [1] – is based on expressing the conditions of conditional clauses within a standard clause. The non-trivial part is to keep the translation linear. This can be achieved by using the conditions as “flags”. For example, consider a conditional iteration clause  $X :-_{\varphi_1, \varphi_2} \psi_1; \psi_2; \chi$ . The corresponding standard iteration clause is

$$X :- (\varphi_1 \wedge \psi_1) \vee (\neg\varphi_1 \wedge ((\varphi_2 \wedge \psi_2) \vee (\neg\varphi_2 \wedge \chi))),$$

which is clearly equivalent and linear in size to the original conditional iteration clause. This translation can be easily generalized for arbitrary conditional iteration clauses. ◀

It is easy to get the following corresponding result for MPMSC from the proof of the previous lemma, recalling that terminal clauses in MPMSC are always of modal depth zero.

► **Corollary 6.** *Given a  $\Pi$ -program of MPMSC of size  $m$ , we can construct a strongly equivalent  $\Pi$ -program of MMSC of size  $\mathcal{O}(m)$  and with the same maximum modal depth of iteration clauses and with terminal clauses of modal depth zero. All diamond operators in the constructed program also appear in the original one.*

We are now ready to prove the following.

► **Theorem 7.** *Given  $\Pi$ ,  $\Delta$  and a  $\Pi$ -program of MPMSC of size  $m$ , we can build a strongly equivalent MPC for  $(\Pi, \Delta)$  of size  $\mathcal{O}(\Delta m + |\Pi|)$  when  $\Delta > 0$  and  $\mathcal{O}(m + |\Pi|)$  when  $\Delta = 0$ .*

**Proof.** We give the proof idea; the full proof is in [1]. We first transform the MPMSC-program to a strongly equivalent MMSC-program (Corollary 6). From that program, we construct an MPC whose state string stores the truth values of head predicates and proposition symbols. The circuit is essentially constructed directly from the inverse tree representations of clauses. Head predicates and proposition symbols in the scope of a diamond will correspond to input gates for bits sent by neighbouring nodes. Moreover, head predicates and propositions not in the scope of a diamond relate to input gates for the home node. In communication round zero, the circuit uses a subcircuit constructed from terminal clauses, and in later rounds, it uses a subcircuit constructed from iteration clauses. ◀

## 5 Linking standard MSC to MPC and MPMSC

To simulate MPMSC (and MMSC) in MSC, we will need to simulate each  $\diamond_i$  with  $\diamond$  only. The following lemma is the key step in the process. In the lemma, note that while the computation time may seem large at first,  $|\Pi_1|$  is typically logarithmic.

► **Lemma 8.** *Given  $\Pi$  and a  $\Pi$ -program of MPMSC of size  $m$  where the maximum subindex of a diamond is  $I$ , we can construct an equivalent  $\Pi$ -program of CMSC of size  $\mathcal{O}(I + |\Pi_1| + m)$ . The computation time is  $\mathcal{O}(2^{|\Pi_1|})$  times the computation time of the MPMSC-program.*

**Proof.** Let us first discuss the key ideas of the proof. The key idea of simulating diamonds  $\diamond_i$  with  $\diamond$  is to scan through the neighbours one by one, in the order given by the IDs. To keep the outputs of our translation small in size, different diamonds  $\diamond_i$  will be “read” in different rounds. For this, we will use, together with IDs, the notion of a *clock*.

Clocks are an essential part in the proof, so let us discuss how they operate. A clock is basically a subprogram controlling head predicates  $M_1, \dots, M_\ell$ , where  $\ell = |\Pi_1|$ . At each node and in each iteration round of a CMSC-program, the truth values of the head predicates  $M_1, \dots, M_\ell$  always define a binary string  $s$  with  $\ell$  bits. While  $s$  changes during computation, different nodes have the same  $s$  at any given time instant. More formally, letting  $s_u(i)$  denote  $s$  at node  $u$  at iteration step  $i$ , we have  $s_u(i) = s_v(i)$  for all  $u$  and  $v$ . In the first iteration step, we have  $s = 0^\ell$ , and then, the string  $s$  goes through all the  $\ell$ -bit strings in lexicographic order. After that, the process starts again from  $0^\ell$ .

The clock string  $s$  is constant for more than a single iteration round of the CMSC-program. There are two reasons for this. Firstly, updating the clock string  $s$  to the lexicographically next string takes some time (and uses some auxiliary head predicates). Secondly, the clock has been designed to help the main program simulate multimodal diamonds  $\diamond_i$  with the single diamond  $\diamond$  of CMSC, and this requires some time. Let us next discuss how the clock string is indeed used.

For each string  $s$ , the main CMSC-program scans through all neighbours at each node. The goal is to find a neighbour whose ID is a precise match with  $s$ . Let  $X_{\text{ID}}$  be a head predicate that becomes true at each node  $u$  precisely at those rounds where the ID of  $u$  matches with  $s$ . Then, at node  $v$ , checking whether some neighbour has an ID matching the current string  $s$  is reduced to checking if  $\diamond X_{\text{ID}}$  holds.

Using the value of  $X_{ID}$  at neighbouring nodes, it is easy to simulate each  $\diamond_i$  with  $\diamond$ , as long as we reserve enough time for scanning through all neighbours of each node. For the full formal details, see [1]. ◀

The next theorem follows immediately from the above Lemma and Lemma 5.

► **Theorem 9.** *Given  $\Pi$  and a  $\Pi$ -program of MPMSC of size  $m$  where the maximum subindex in a diamond is  $I$ , we can construct an equivalent  $\Pi$ -program of MSC of size  $\mathcal{O}(I + |\Pi_1| + m)$ . The computation time is  $2^{\mathcal{O}(|\Pi_1|)}$  times the computation time of the MPMSC-program.*

## 5.1 A normal form for MSC

A program of MSC[1] is a program of MSC where the modal depth of terminal (respectively, iteration) clauses is zero (resp., at most one). This normal form of MSC is essentially used as the tool when translating a program of MSC to MPMSC and ultimately to MPC. We begin with the following lemma that shows we can force the modal depth of each terminal clause to zero.

► **Lemma 10.** *For every  $\Pi$ -program  $\Lambda$  of MSC, there exists an equivalent  $\Pi$ -program of MSC where the modal depth of terminal clauses is zero. The size of the program is linear in the size of  $\Lambda$  and the computation time is linear in the computation time of  $\Lambda$ .*

**Proof.** We sketch the proof; for the full proof, see [1]. The proof is based on (1) using CMSC suitably in order to modify terminal clauses so that their diamonds become part of iteration clauses and (2) then translating CMSC to MSC. ◀

We then show that the modal depth of iteration clauses can be reduced to one.

► **Theorem 11.** *For every  $\Pi$ -program  $\Lambda$  of MSC, there exists an equivalent  $\Pi$ -program of MSC[1]. The size of the MSC[1]-program is linear in the size of  $\Lambda$  and the computation time of the program is  $\mathcal{O}(\max(1, \text{md}(\Lambda)))$  times the computation time of  $\Lambda$ .*

**Proof.** We sketch the proof; for the full proof, see [1]. We first transform the original MSC-program to one where the modal depth of the terminal clauses is zero by Lemma 10. Then we use CMSC to replace each subschema of type  $\diamond\psi$  with a fresh head predicate  $X_{\diamond\psi}$  such that in the thereby obtained program, the modal depth of each iteration clause is at most 1. Finally, we translate CMSC to MSC by Lemma 5. ◀

## 5.2 Linking MSC and MPCs

We are now ready to link MSC to MPCs. In Section 4.1 we proved Theorem 4 that shows we can translate MPCs to strongly communication equivalent MPMSC-programs of size linear in the size of the MPC. On the other hand, Theorem 9 shows that we can translate any MPMSC-program to an equivalent program of MSC. We get the following theorem.

► **Theorem 12.** *Given an MPC for  $(\Pi, \Delta)$ , we can construct an equivalent  $\Pi$ -program of MSC. For a constant bound  $c$  for the fan-in of MPCs, the size of the program is linear in the size of the circuit. The computation time is  $\mathcal{O}(d + 2^{|\Pi_1|})$  times the computation time of the MPC, where  $d$  is the depth of the MPC.*

Theorem 7 showed that we can translate an MPMSC-program to a strongly equivalent MPC. Theorem 11 showed how to translate an MSC-program to a strongly equivalent MSC[1]-program, implying that translating an MSC-program to an MPMSC-program can be done without blowing up program size too much. These results directly imply the following.

► **Theorem 13.** *Given  $\Pi$ ,  $\Delta$  and a  $\Pi$ -program of MSC of size  $m$ , there exists an equivalent MPC for  $(\Pi, \Delta)$  of size  $\mathcal{O}(\Delta m + |\Pi|)$  when  $\Delta > 0$  and  $\mathcal{O}(m + |\Pi|)$  when  $\Delta = 0$ . The computation time is  $\mathcal{O}(\max(1, d))$  times the computation time of the MSC-program, where  $d$  is the modal depth of the MSC-program.*

By the above results, we observe that problems in  $\text{DCC}_{\Delta}^c[\log n]$  can be alternatively described with sequences of MSC-programs.

Finally, we note that Theorem 4 is one of our main results. It reminds us that communication time is indeed a different concept than computation time.

## 6 Brief notes on graph coloring

As proof-of-concept for this article, we briefly and informally discuss the Cole-Vishkin algorithm [5], a fundamental method used in distributed graph coloring. The CV-algorithm takes advantage of a phenomenon whereby it is possible to logarithmically reduce the size of a binary string by replacing it with a binary encoding of one of its positions. By iterating this technique, it is possible to reduce the size of an  $n$ -size string down to three in  $\mathcal{O}(\log^*(n))$  iterations. Applied as a distributed algorithm to an  $n$ -coloring in an oriented tree or forest, it is possible to reduce the number of colors to single digits in  $\mathcal{O}(\log^*(n))$  communication rounds [6]. By extension, Barenboim and Elkin [3] show a number of ways this can be combined with other simpler iterative algorithms to produce fast  $(\Delta + 1)$ -color reduction algorithms, i.e. algorithms that reduce the number of colors from the size of a graph down to its maximum degree plus one, which is optimal in the worst-case scenario.

While the communication time of these algorithms has been studied before, little is generally understood about the duration of their local (node-internal) computation and the necessary program length required to formally express them.

In the full preprint version of this paper [1] (available online), we prove that given a bound for the degree  $\Delta$  of a graph, the CV-algorithm and a broader simple  $(\Delta + 1)$ -color reduction algorithm can be expressed with a program of MPMSC with size logarithmic in the number of nodes. Additionally, the expression is uniform for all degree bounds. In other words, for any  $\Delta$ , the Cole-Vishkin algorithm (and the associated  $(\Delta + 1)$ -color reduction algorithm) can be expressed in a compact way in MSC and thus also in the related distributed computing class where MPCs are from  $\text{NC}^1$ . The following theorem is obtained as a result.

► **Theorem 14.** *Given a bounded-degree graph with at most  $n$  nodes, there exists an MPMSC-program of size  $\mathcal{O}(\log(n))$  that defines a  $(\Delta + 1)$ -coloring for the graph. The computation time is  $\mathcal{O}(\log(n) \log(\log(n)) \log^*(n))$  of which  $\log^*(n) + \mathcal{O}(1)$  are global communication rounds.*

By Theorems 14 and 9, we get a program of MSC of size  $\mathcal{O}(|\Pi_1| + \log(n))$  with an increase in computation time by a factor of  $2^{\mathcal{O}(|\Pi_1|)}$ . While the computation time may seem large, note that  $|\Pi_1|$  is typically logarithmic. We emphasize that the computation and communication times of a program are very different concepts and the former will usually dwarf the latter.

## 7 Conclusion

We have characterized distributed computation via circuits in terms of the logic MSC. The translations lead to only polynomial increase in size, and in the constant-degree scenario, the increase is only linear. In the future, we aim to expand these studies to concern models with weights, pushing the approach closer to work on neural networks.

---

**References**

---

- 1 Veeti Ahvonen, Damian Heiman, Lauri Hella, and Antti Kuusisto. Descriptive complexity for distributed computing with circuits. *CoRR*, abs/2303.04735v1, 2023. doi:10.48550/arXiv.2303.04735.
- 2 Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020.
- 3 Leonid Barenboim and Michael Elkin. Distributed graph coloring. *Synthesis Lectures on Distributed Computing Theory*, 11, 2013.
- 4 Benedikt Bollig, Patricia Bouyer, and Fabian Reiter. Identifiers in registers – Describing network algorithms with logic. *CoRR*, abs/1811.08197, 2018.
- 5 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.
- 6 Andrew Goldberg, Serge Plotkin, and Gregory Shanon. Parallel symmetry-breaking in sparse graphs. *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324, 1987.
- 7 Martin Grohe. The logic of graph neural networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–17. IEEE, 2021.
- 8 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12*, pages 185–194. ACM, 2012.
- 9 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Comput.*, 28(1):31–53, 2015.
- 10 Stefanie Jegelka. Theory of graph neural networks: Representation and learning. *arXiv preprint*, 2022. arXiv:2204.07697.
- 11 Antti Kuusisto. Modal Logic and Distributed Message Passing Automata. In *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 452–468, 2013.
- 12 Tuomo Lempiäinen. *Logic and Complexity in Distributed Computing*. PhD thesis, Aalto University, 2019.
- 13 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
- 14 Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint*, 2019. arXiv:1907.03199.
- 15 Fabian Reiter. Asynchronous distributed automata: A characterization of the modal mu-fragment. In I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.
- 16 Fabian Reiter. *Distributed Automata and Logic. (Automates Distribués et Logique)*. PhD thesis, Sorbonne Paris Cité, France, 2017.
- 17 Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.



# Solving Irreducible Stochastic Mean-Payoff Games and Entropy Games by Relative Krasnoselskii–Mann Iteration

Marianne Akian 

INRIA and CMAP, École polytechnique, IP Paris, CNRS, France

Stéphane Gaubert 

INRIA and CMAP, École polytechnique, IP Paris, CNRS, France

Ulysse Naepels 

École polytechnique, IP Paris, France

Basile Terver 

École polytechnique, IP Paris, France

---

## Abstract

We analyse an algorithm solving stochastic mean-payoff games, combining the ideas of relative value iteration and of Krasnoselskii–Mann damping. We derive parameterized complexity bounds for several classes of games satisfying irreducibility conditions. We show in particular that an  $\epsilon$ -approximation of the value of an irreducible concurrent stochastic game can be computed in a number of iterations in  $O(|\log \epsilon|)$  where the constant in the  $O(\cdot)$  is explicit, depending on the smallest non-zero transition probabilities. This should be compared with a bound in  $O(\epsilon^{-1}|\log(\epsilon)|)$  obtained by Chatterjee and Ibsen-Jensen (ICALP 2014) for the same class of games, and to a  $O(\epsilon^{-1})$  bound by Allamigeon, Gaubert, Katz and Skomra (ICALP 2022) for turn-based games. We also establish parameterized complexity bounds for entropy games, a class of matrix multiplication games introduced by Asarin, Cervelle, Degorre, Dima, Horn and Kozyakin. We derive these results by methods of variational analysis, establishing contraction properties of the relative Krasnoselskii–Mann iteration with respect to Hilbert’s semi-norm.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Algorithmic game theory

**Keywords and phrases** Stochastic mean-payoff games, concurrent games, entropy games, relative value iteration, Krasnoselskii–Mann fixed point algorithm, Hilbert projective metric

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.10

**Related Version** *Extended Version*: <https://arxiv.org/pdf/2305.02458.pdf>

**Acknowledgements** We thank the reviewers for helpful comments.

## 1 Introduction

### 1.1 Motivation and context

Stochastic mean-payoff games are a fundamental class of zero-sum games, appearing in various guises. In *turn-based* games, two players play sequentially, alternating moves, or choices of an action, being aware of the previous decision of the other player. Turn-based games with mean-payoff and finite state and action spaces are among the unsettled problems in complexity theory: they belong to the complexity class  $\text{NP} \cap \text{coNP}$  [14, 40] but are not known to be polynomial-time solvable. We refer the reader to the survey [7] for more information on the different classes of turn-based games. In contrast, in *concurrent games*, at each stage, the two players choose simultaneously one action, being unaware of the choice of the other player at the same stage. Turn-based games are equivalent to a subclass of concurrent games (in which in each state, one of the two players is a dummy). The existence



© Marianne Akian, Stéphane Gaubert, Ulysse Naepels, and Basile Terver;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 10; pp. 10:1–10:15

Leibniz International Proceedings in Informatics



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

of the value for concurrent stochastic mean-payoff games is a celebrated result of Mertens and Neyman [28]. This builds on earlier results by Bewley and Kohlberg, connecting mean-payoff concurrent games with discounted concurrent games, by making the discount factor tend to 1, see [11]. Concurrent games are hard to solve exactly: the value is an algebraic number whose degree may be exponential in the number of states [21]. Moreover, concurrent reachability games are square-root sum hard [16].

Another class consists of *entropy games*, introduced by Asarin, Cervelle, Degorre, Dima, Horn and Kozyakin as an interesting category of “matrix multiplication games” [8]. Entropy games capture a variety of applications, arising in risk sensitive control [23, 6], portfolio optimization [3], growth maximization and population dynamics [36, 33, 32, 39]. Asarin et al. showed that entropy games belong to the class  $\text{NP} \cap \text{coNP}$ , showing an analogy with turn based games. In [1], Akian, Gaubert, Grand-Clément and Guillaud showed that entropy games are actually special cases of stochastic mean-payoff games, in which action spaces are infinite sets (simplices), and payments are given by Kullback-Leibler divergences.

A remarkable subclass of stochastic mean-payoff games arises when imposing *ergodicity* or *irreducibility* conditions. Such conditions entail that the value of the game is independent of the initial state. The simplest condition of this type requires that every pair of policies (Markovian stationary strategies) of the two players induces an irreducible Markov chain. Then, the solution of the game reduces to solving a nonlinear eigenproblem of the form  $T(u) = \lambda e + u$ , in which  $u \in \mathbb{R}^n$  is a non-linear eigenvector,  $\lambda$  is a non-linear eigenvalue, which provides the value of the mean-payoff game,  $e$  is the unit vector of  $\mathbb{R}^n$ , and  $T$  is a self-map of  $\mathbb{R}^n$ , the dynamic programming operator of the game, which we shall refer to as the “Shapley” operator. In fact, Shapley originally introduced a variant of this operator, adapted to the discounted case [34]. The undiscounted mean-payoff case was subsequently considered by Gillette [20]. We refer the reader to [29, 31] for background on Shapley operators and on the “operator approach” to games, and to [2] for a discussion of the non-linear eigenproblem.

In the one-player case, White [38] introduced *relative value iteration*, which consist in fixed point iterations up to additive constants  $\lambda_k \in \mathbb{R}$ , i.e.  $x_{k+1} = T(x_k) - \lambda_k e$ . This solves the non-linear eigenproblem  $T(u) = \lambda e + u$  under a primitivity assumption. However, this assumption appears to be too restrictive in the light of the classical Krasnoselskii–Mann algorithm [25, 27], which allows one to find a fixed point of a nonexpansive self-map  $T$  of a finite dimensional normed space, by constructing the “damped” sequence  $x_{k+1} = (1 - \theta)T(x_k) + \theta x_k$ , where  $0 < \theta < 1$ . Indeed, it was proposed in [19] to apply this algorithm to the non-linear eigenproblem  $T(u) = \lambda e + u$ , thought of as a fixed point problem in the quotient vector space  $\mathbb{R}^n / \mathbb{R}e$ . We will refer to this algorithm as the *relative Krasnoselskii–Mann value iteration*. An error bound in  $O(1/\sqrt{k})$  was derived in [19] for this algorithm, as a consequence of a general theorem of Baillon and Bruck [10], and the existence of an asymptotic geometric convergence rate was established in a special case. This left open the question of obtaining stronger iteration complexity bounds, in a “white box model”, for specific classes of stochastic mean-payoff games.

## 1.2 Contribution

We apply the relative Krasnoselskii–Mann value iteration algorithm to deduce complexity bounds for several classes of stochastic games. We consider in particular *unichain* concurrent stochastic mean-payoff games, in which every pair of policies of the two players induces a unichain transition matrix (i.e., a stochastic matrix with a unique final class). We define  $p_{\min}$  to be the smallest non-zero off-diagonal transition probability in the model. Corollary 20 shows that the relative Krasnoselskii–Mann iteration yields an  $\epsilon$ -approximation of the value

of the game, after  $C|\log \epsilon|$  iterations. The factor  $C$  is exponential in the bit-size of the input, it has an essential term of the form  $k\theta^{-k}$ , in which  $k \leq n$  is a certain “unichain index”, which is equal to 1 if all the transition probabilities are positive,  $\theta = p_{\min}/(1 + p_{\min})$ , and  $n$  denotes the number of states. Then, we consider the special case of unichain turn-based games, with rational transition probabilities whose denominator divides  $M$ . Theorem 23 shows that optimal policies can be obtained after a number of iterations of order  $M^k$ . The main tool is Theorem 19, which shows that a suitable iterate of the Shapley operator of a unichain concurrent game is a contraction in Hilbert’s seminorm. This theorem is proved using techniques of variational analysis, in particular we use a classical result of Mills [30], characterizing the directional derivative of the value of a matrix game, and properties of nonsmooth semidifferentiable maps.

Finally, we introduce a variant of the relative Krasnoselkii–Mann algorithm, adapted to entropy games. Theorem 26 shows that an irreducible entropy game can be solved exactly in a time of order  $(1 + \mathcal{A}/\underline{m})^k$  where  $k \leq n$  is a certain “irreducibility index”,  $\underline{m} \geq 1$  is the smallest multiplicity of an off-diagonal transition, and  $\mathcal{A}$  is a measure of the *ambiguity* of the game. In particular, we have  $W \leq \mathcal{A} \leq n^{1-1/n}W$  where  $W$  is the maximal multiplicity of a transition. The proof exploits the Birkhoff–Hopf theorem, which states that a positive matrix is a contraction in Hilbert’s projective metric.

The proofs of the present results can be found in the extended version of this article [4].

### 1.3 Related work

The algorithmic approach of stochastic mean-payoff games satisfying irreducibility conditions goes back to the work of Hoffman and Karp [22], applying policy iteration. Chatterjee and Ibsen-Jensen [13] studied concurrent stochastic mean-payoff games, under appropriate conditions of ergodicity. They showed in particular that the problem of approximation of the value is in FNP, and that this approximation problem, restricted to turn-based ergodic games, is at least as hard as the decision problem for simple stochastic games. They also showed that value iteration provides an  $\epsilon$ -approximation of the value of a concurrent stochastic game satisfying an irreducibility condition in  $O(\tau\epsilon^{-1}|\log \epsilon|)$  iterations, where  $\tau$  denotes a bound of the passage time between any two states under an arbitrary strategy, see Theorem 18, *ibid.* A recent “universal bound” on value iteration by Allamigeon, Gaubert, Katz and Skomra [5, Th. 13] entails an improvement of this bound to  $O(\tau\epsilon^{-1})$ . Corollary 20 further improves this bound to get  $C|\log \epsilon|$ . However, the later result requires an unichain assumption, whereas the assumption of [5, Th. 13] is milder.

The question of computing the value of a concurrent discounted stochastic game has been studied by Hansen, Koucký, Lauritzen, Miltersen and Tsigaridas in [21], who showed, using semi-algebraic geometry techniques, that an  $\epsilon$ -approximation of the value of a general concurrent game can be obtained in polynomial time if the number  $n$  of states is fixed. The exponent of the polynomial is of order  $O(n)^{n^2}$  and it was remarked in [21] that “getting a better dependence on  $n$  is a very interesting open problem”. Boros, Gurvich, Elbassioni and Makino considered the notion of  $\epsilon$ -ergodicity of a concurrent mean-payoff game, requiring that the mean-payoff of two initial states differ by at most  $\epsilon$ . They provided a potential-reduction algorithm allowing one to decide  $\epsilon$ -ergodicity, and to get an  $\epsilon$ -approximation of the value, with a dependence in  $\epsilon$  of order  $\epsilon^{-O(2^{2n}n \max(|A|, |B|))}$ , see [12]. Attia and Oliu-Barton developed in [9] a bisection algorithm, with a complexity bound polynomial in  $|\log \epsilon|$  and in  $|A|^n$  and  $|B|^n$  where  $A, B$  are the action spaces. In contrast to these three works, our approach only applies to the subclass of *unichain* concurrent games, but its complexity has a better dependence in the number of states; in particular, the exponents in our bound is at

most  $n$ , and the execution time grows only polynomially with the numbers of actions  $|A|$  and  $|B|$ . Moreover, our approach applies more generally to infinite (compact) action spaces (we only need an oracle evaluating the value of a possibly infinite matrix game up to a given accuracy).

The analysis of relative value iteration, using contraction techniques, goes back to the work of Federguen, Schweitzer and Tijms [17], dealing with the one-player and finite action spaces case, under a primitivity condition. The novelty here is the analysis of the concurrent two-player case, as well as the analysis of the effect of the Krasnoselskii–Mann damping, allowing one to replace earlier primitivity conditions by a milder unichain condition. Moreover, even in the one-player case, our formula for the contraction rate given in Theorem 19 improves the one of [17].

Our results of Section 9 dealing with entropy games are inspired by the series of works [8, 1, 5]. The subclass of “Despot-free” entropy games can be solved in polynomial time [1], and it is an open question whether general entropy games can be solved in polynomial time. The approach of [5] entails that one can get an  $\epsilon$ -approximation of the value of an entropy game in  $O(\epsilon^{-1})$  iterations, where the factor in the  $O(\cdot)$  is exponential in the parameters of the game. This bound is refined here to  $O(|\log \epsilon|)$ , in which the factor in the  $O(\cdot)$  depends on a measure of “ambiguity” – but our approach requires an irreducibility assumption.

## 2 Preliminary results on Shapley operators

Let  $n$  be an integer. A map  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is said to be *order-preserving* when:  $\forall x, y \in \mathbb{R}^n, x \leq y \implies T(x) \leq T(y)$ , where  $\leq$  denotes the standard partial order of  $\mathbb{R}^n$ . It is *additively homogeneous* when:  $\forall x \in \mathbb{R}^n, \forall \lambda \in \mathbb{R}, T(x + \lambda e) = T(x) + \lambda e$  where  $e$  is the vector of  $\mathbb{R}^n$  having 1 in each coordinate.

► **Definition 1.** *A map  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is an (abstract) Shapley operator if it is order-preserving and additively homogeneous.*

We will justify the terminology “Shapley operator” in the next section, where we give concrete examples, arising as dynamic programming operators of different classes of zero-sum repeated games. We set  $[n] := \{1, \dots, n\}$ . For any  $x \in \mathbb{R}^n$ , we denote  $\mathbf{t}(x) := \max_{i \in [n]} x_i$  and  $\mathbf{b}(x) := \min_{i \in [n]} x_i$  (read “top” and “bottom”). We define the *Hilbert’s seminorm* of  $x$  by:  $\|x\|_H = \mathbf{t}(x) - \mathbf{b}(x)$ . Since  $\|x\|_H = 0$  iff  $x \in \mathbb{R}e$ , we get that  $\|\cdot\|_H$  is actually a norm on the quotient vector space  $\mathbb{R}^n/\mathbb{R}e$ . We also notice that  $\|x\|_\infty = \inf\{\lambda \in \mathbb{R}_+ \mid -\lambda e \leq x \leq \lambda e\}$  and  $\|x\|_H = \inf\{\beta - \alpha \in \mathbb{R}_+ \mid \alpha, \beta \in \mathbb{R}, \alpha e \leq x \leq \beta e\}$ . It is easy to show, thanks to these expressions, that a Shapley operator  $T$  is non-expansive (i.e., 1-Lipschitz) for  $\|\cdot\|_H$  and for  $\|\cdot\|_\infty$ . Then, it induces a self-map  $\bar{T}$  on the quotient vector space  $\mathbb{R}^n/\mathbb{R}e$ , sending the equivalence class  $x + \mathbb{R}e$  to  $T(x) + \mathbb{R}e$ , and which is non-expansive.

► **Definition 2.** *We define the escape rate  $\chi(T)$  of a Shapley operator  $T$  as  $\lim_{k \rightarrow \infty} k^{-1}T^k(v)$ , where  $v$  is an arbitrary vector in  $\mathbb{R}^n$ . The lower and upper escape rates are defined respectively by  $\underline{\chi}(T) = \lim_{k \rightarrow \infty} k^{-1}\mathbf{b}(T^k(v))$  and  $\bar{\chi}(T) = \lim_{k \rightarrow \infty} k^{-1}\mathbf{t}(T^k(v))$ .*

Since  $T$  is nonexpansive in the sup-norm, the existence and the values of these limits are independent of the choice of  $v \in \mathbb{R}^n$ . In general, the escape rate  $\chi(T) = \lim_{k \rightarrow \infty} k^{-1}T^k(v)$  may not exist, but a subadditive argument shows that the lower and upper escape rates always exist, see e.g. [18]. A fundamental tool to establish the existence of the escape rate is to consider the following ergodic equation.

► **Definition 3.** *We say that the ergodic equation has a solution when there exists  $\lambda \in \mathbb{R}$  and  $u \in \mathbb{R}^n$  such that  $T(u) = \lambda e + u$ .*

► **Observation 4.** *If the above ergodic equation is solvable, then  $\chi(T) = \lambda e$ . More generally, if  $\alpha e + v \leq T(v) \leq \beta e + v$  for some  $v \in \mathbb{R}^n$  and  $\alpha, \beta \in \mathbb{R}$ , then  $\alpha \leq \underline{\chi}(T) \leq \bar{\chi}(T) \leq \beta$ .*

**Proof.** By an immediate induction, and as  $T$  is order-preserving and additively homogeneous we have :  $k\alpha e + v \leq T^k(v) \leq k\beta e + v$ . Then,  $k\alpha + \mathbf{b}(v) \leq \mathbf{b}(T^k(v)) \leq \mathbf{t}(T^k(v)) \leq k\beta + \mathbf{t}(v)$ . Dividing by  $k$  and letting  $k$  tend to infinity, we obtain the second statement. ◀

We are inspired by the following observation from fixed point theory.

► **Observation 5.** *Suppose now that  $T^q$  is  $\gamma$ -contraction in Hilbert's seminorm  $\|\cdot\|_H$ , for some  $q \geq 1$  and  $0 < \gamma < 1$ . Then, the ergodic equation is solvable.*

Shapley operators include (finite dimensional) Markov operators, which are of the form  $T(x) = Mx$ , where  $M$  is a  $n \times n$  stochastic matrix (meaning that  $M$  has nonnegative entries and row sums one). In this case, an exact formula is known for the contraction rate. In fact, one can consider the operator norm of  $M$ , thought of as a linear map acting on the quotient vector space  $\mathbb{R}^n / \mathbb{R}e$ ,  $\|M\|_H = \sup_{u \notin \mathbb{R}e} \frac{\|Mu\|_H}{\|u\|_H}$ .

► **Theorem 6** (Corollary of [15]).  $\|M\|_H = \delta(M) := 1 - \min_{1 \leq i < j \leq n} \{ \sum_{k \in [n]} \min(M_{ik}, M_{jk}) \}$ .

The term  $\delta(M)$  is known as *Dobrushin ergodicity coefficient*.

### 3 Two classes of zero-sum two-player repeated games

We next recall the definition and basic properties of two classes of zero-sum two-player games with finite state spaces. More details can be found in [29] for stochastic games and in [8, 1] for entropy games.

#### 3.1 Concurrent repeated zero-sum stochastic two-player games

We assume that the state space is equal to  $[n] = \{1, \dots, n\}$ . We call the two players “Min” and “Max”. The game is specified by the following data. For every state  $i \in [n]$ , we are given two non-empty compact sets  $A(i)$  and  $B(i)$ , representing the admissible actions of players Min and Max, respectively. For every  $i \in [n]$  and every choice of actions  $(a, b) \in A(i) \times B(i)$ , we are given a real number  $r_i^{ab}$ , representing an instantaneous payment, and a stochastic vector  $P_i^{a,b} = (P_{ij}^{a,b})_{j \in [n]}$ , meaning that  $P_{ij}^{a,b} \geq 0$  and that  $\sum_{j \in [n]} P_{ij}^{a,b} = 1$ . We assume that the functions  $(a, b) \mapsto r_i^{ab}$  and  $(a, b) \mapsto P_i^{a,b}$  are continuous.

The concurrent game is played in successive stages, starting from a known initial state  $i_0$  at stage 0. We denote by  $a_k$  and  $b_k$  the actions selected by Players Min and Max at stage  $k$ , respectively, and by  $i_k$  the state at this stage. The history until stage  $k$  consists of the sequence  $H_k = ((i_\ell, a_\ell, b_\ell)_{0 \leq \ell < k}, i_k)$ . A randomized strategy of Player Min (resp. Max) is a collection of measurable functions assigning to every history  $H_k$  a probability measure  $\alpha_k$  (resp.  $\beta_k$ ) on the compact set  $A(i_k)$  (resp.  $B(i_k)$ ). At stage  $k$ , being informed of the history  $H_k$  up to this stage, Player Min draws a random action  $a_k$  according to the probability measure  $\alpha_k$ , and similarly, Player Max draws a random action  $b_k$  according to the probability measure  $\beta_k$ . Then, Player Min makes to Player Max an instantaneous payment of  $r_{i_k}^{a_k, b_k}$ , and the next state  $i_{k+1}$  is drawn randomly according to the probability measure  $(P_{i_k, j}^{a_k, b_k})_{j \in [n]}$  on the state space  $[n]$ , i.e., the conditional probability that  $i_{k+1} = j$ , given the history  $H_k$  and actions  $a_k, b_k$ , is given by  $P_{i_k, j}^{a_k, b_k}$ . We shall say that a strategy is *pure* or *deterministic* if the action of the player is chosen as a deterministic function of the history. We denote by  $\sigma_k$  (resp.  $\tau_k$ ) the strategy of Player Min (resp. Max) at stage  $k$ , and denote by  $\sigma$  and  $\tau$

## 10:6 Relative Krasnoselskii–Mann Iteration

the sequences  $(\sigma_k)_{k \geq 0}$  and  $(\tau_k)_{k \geq 0}$ . In this way, to any initial state  $i_0 \in [n]$  and any pair of strategies  $(\sigma, \tau)$  of the two players is associated the infinite random sequence  $(i_k, a_k, b_k)_{k \geq 0}$ . We denote by  $\mathbb{E}_{i_0}^{\sigma, \tau}$  the expectation operator with respect to this process.

We shall consider special classes of strategies. We denote by  $\Delta_X$  the set of probability measures on a compact set  $X$ . A *randomized policy* of Player Min is a map  $\alpha : [n] \rightarrow \cup_{i \in [n]} \Delta_{A(i)}$ ,  $i \mapsto \alpha_i$ . For each  $i \in [n]$ ,  $d\alpha_i(a)$  determines the probability an action  $a \in A_i$  is chosen, according to this policy. Thus, the set of randomized policies of Min,  $\Pi_{\mathbb{R}}^{\text{Min}}$ , can be identified to  $\prod_{i \in [n]} \Delta_{A(i)}$ . A policy of Min is said to be *pure* if for all  $i \in [n]$ ,  $\alpha_i$  is a Dirac measure, so that  $d\alpha_i = \delta_{a_i}$  for some  $a_i \in A_i$ . Such a policy prescribes to play the deterministic action  $a_i$  when in state  $i$ . Therefore, a pure policy is uniquely specified by the map  $i \mapsto a_i$ . This allows us to identify the set of pure policies, denoted by  $\Pi_{\mathbb{P}}^{\text{Min}}$ , to the product  $\prod_{i \in [n]} A_i$ . A randomized *Markovian* strategy  $\sigma$  of Player Min is a strategy such that the decision prescribed by  $\sigma_k$  depends only on the current state  $i_k$ . In other words, it is obtained by selecting, at each time step, a randomized policy of Player Min, and playing the action according to this policy. A Markovian strategy is *pure* if only pure policies are selected. It is *stationary* if the same policy is applied at every time step  $k$ . In this way, a pure (resp. randomized) Markovian stationary strategy can be identified to a pure (resp. randomized) policy. We shall use the same notation and terminology for Player Max, mutatis mutandis. In particular, we denote by  $\Pi_{\mathbb{R}}^{\text{Max}}$  and  $\Pi_{\mathbb{P}}^{\text{Max}}$  the sets of randomized and pure policies of Player Max. We shall also denote by  $\Pi_{\mathbb{P}} = \Pi_{\mathbb{P}}^{\text{Min}} \times \Pi_{\mathbb{P}}^{\text{Max}}$  and  $\Pi_{\mathbb{R}} = \Pi_{\mathbb{R}}^{\text{Min}} \times \Pi_{\mathbb{R}}^{\text{Max}}$  the spaces of pairs of policies.

Given an initial state  $i_0$  and a pair of strategies  $(\sigma, \tau)$  of the two players, the expected payment received by Player Max in horizon  $N$  is defined by

$$J_{i_0}^N(\sigma, \tau) := \mathbb{E}_{i_0}^{\sigma, \tau} \left[ \sum_{k=0}^{N-1} r_{i_k}^{a_k, b_k} \right] .$$

We shall denote by  $J^N(\sigma, \tau)$  the vector of  $\mathbb{R}^n$  with the above  $i_0$  entry, for each  $i_0 \in [n]$ . The finite horizon game has a value  $v^N \in \mathbb{R}^n$  and has a pair of optimal (randomized) strategies  $(\sigma^*, \tau^*)$ , meaning that

$$J^N(\sigma^*, \tau) \leq v^N = J^N(\sigma^*, \tau^*) \leq J^N(\sigma, \tau^*) , \quad (1)$$

for all pairs  $(\sigma, \tau)$  of strategies, see [29]. Moreover, one can choose the pair of optimal strategies  $(\sigma^*, \tau^*)$  to be Markovian, that is  $(\sigma_k^*, \tau_k^*) \in \Pi_{\mathbb{R}}$  for all  $k \leq N$  (but it generally depends on  $k$  and  $N$ ). These optimal strategies can be obtained by using the dynamic programming equation of the game, as follows.

For any  $i, j \in [n]$ ,  $\alpha_i \in \Delta_{A(i)}$  and  $\beta_i \in \Delta_{B(i)}$ , let us denote

$$r_i^{\alpha_i, \beta_i} = \int_{A(i) \times B(i)} r_i^{a, b} d\alpha_i(a) d\beta_i(b) \quad \text{and} \quad P_{i, j}^{\alpha_i, \beta_i} = \int_{A(i) \times B(i)} P_{i, j}^{a, b} d\alpha_i(a) d\beta_i(b) . \quad (2)$$

This extends the functions  $(a, b) \mapsto r_i^{a, b}$  and  $(a, b) \mapsto P_{i, j}^{a, b}$  from  $A(i) \times B(i)$  to  $\Delta_{A(i)} \times \Delta_{B(i)}$ . We then define the Shapley operator  $T$  of the concurrent game as the map  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  such that

$$T_i(v) = \min_{\alpha_i \in \Delta_{A(i)}} \max_{\beta_i \in \Delta_{B(i)}} \left( r_i^{\alpha_i, \beta_i} + \sum_{j \in [n]} P_{i, j}^{\alpha_i, \beta_i} v_j \right), \quad \text{for } i \in [n], v \in \mathbb{R}^n . \quad (3)$$

Note that in the above expression the infimum and supremum commute, owing to the compactness of action spaces, and continuity assumptions on the functions  $(a, b) \mapsto r_i^{a, b}$  and  $(a, b) \mapsto P_{i, j}^{a, b}$  (this follows from Sion's minimax theorem). Moreover, the operator  $T$  satisfies the properties of Definition 1.

Then, the value of the concurrent game in finite horizon is obtained from the recurrence equations:  $v^0 = 0$ ,  $v^N = T(v^{N-1})$ . Moreover, optimal strategies of the game when the remaining time is  $k < N$  (or at stage  $N - k$ ) are obtained by choosing optimal policies  $\alpha$  and  $\beta$  with respect to the vectors  $v^k$ , that is such that  $\alpha_i$  and  $\beta_i$  are optimal in the expression of  $T_i(v^k)$  in (3).

We now describe the *mean-payoff game*, which is obtained by considering the Cesaro limit of the payoff as the horizon  $N$  tends to infinity. More precisely, we set:

$$\chi_{i_0}^+(\sigma, \tau) := \limsup_{N \rightarrow \infty} N^{-1} J_{i_0}^N(\sigma, \tau) \quad \chi_{i_0}^-(\sigma, \tau) := \liminf_{N \rightarrow \infty} N^{-1} J_{i_0}^N(\sigma, \tau) .$$

We shall say that the game with mean-payoff has a value  $\chi^* \in \mathbb{R}^n$  if for all  $\varepsilon > 0$ , there exists strategies  $\sigma^\varepsilon, \tau^\varepsilon$  of the two players which are  $\varepsilon$ -optimal, meaning that for every strategies  $\sigma$  and  $\tau$ ,  $-\varepsilon e + \chi^+(\sigma^\varepsilon, \tau) \leq \chi^* \leq \chi^-(\sigma, \tau^\varepsilon) + \varepsilon e$ . Mertens and Neyman [28], building on a result of Bewley and Kohlberg [11], showed that when the action spaces  $A(i)$  and  $B(i)$  are finite, the mean-payoff game has a value (actually, in a stronger *uniform* sense). Moreover, the value coincides with the escape rate of the Shapley operator, i.e.,  $\chi^* = \lim_k T^k(0)/k$ . A counter-example of Vigerel shows that these properties do not carry over to the case of general compact action spaces [37].

One particular case that will interest us is when the ergodic equation is solvable, that is when there exists  $\lambda \in \mathbb{R}$  and  $v \in \mathbb{R}^n$  such that  $T(v) = \lambda e + v$ . In that case,  $\chi^* = \lambda e$  and there exists optimal randomized strategies for the two players which are both Markovian and stationary. Such a pair of strategies is obtained by choosing a pair  $(\alpha, \beta)$  of policies such that  $\alpha$  and  $\beta$  achieve the minimum and the maximum, respectively, in the expression of  $T(v)$  in (3). We shall see that the ergodic equation is always solvable under a unichain condition, even in the case of compact action spaces (Theorem 11).

A remarkable subclass of concurrent games consists of *turn-based* games. Then, the actions spaces  $A(i)$  and  $B(i)$  are required to be *finite*, and for every state  $i \in [n]$ , we assume that either  $A(i)$  or  $B(i)$  is a singleton. In other words, there is a bipartition  $[n] = I_{\text{Min}} \uplus I_{\text{Max}}$  of the set of states, so that in every state  $i \in I_{\text{Min}}$  (resp.  $I_{\text{Max}}$ ), Min (resp. Max) is the only player who has to take a decision. Then, the Shapley operator of the game reduces to  $T_i(x) = \min_{a \in A(i)} \max_{b \in B(i)} (r_i^{a,b} + \sum_j P_{i,j}^{a,b} x_j)$ , for  $i \in [n]$ , where again the min and max commute, because in every  $i \in [n]$ , either the min or the max is taken over a set reduced to a singleton. When the ergodic equation  $T(v) = \lambda e + v$  of a turn-based game is solvable, one obtains *pure* optimal policies in the mean-payoff game, by selecting actions that achieve the minimum and the maximum in each coordinate  $[T(v)]_i$  with  $i \in [n]$ . More generally, the existence of pure optimal policies for turn-based mean-payoff stochastic games was shown by Liggett and Lippman [26]. An illustrative example is given in Appendix A.

### 3.2 Entropy games

Entropy games were introduced in [8]. We use here the slightly more general model of [1, 5], to which we refer for background. An *entropy game* is a turn-based game played on a (finite) digraph  $(\mathcal{V}, \mathcal{E})$ , with two players, called “Despot” and “Tribune”, and an additional non-deterministic player, called “People”. We assume the set of vertices  $\mathcal{V}$  has a non-trivial partition:  $\mathcal{V} = \mathcal{V}_D \uplus \mathcal{V}_T \uplus \mathcal{V}_P$ . Players Despot, Tribune, and People control the states in  $\mathcal{V}_D$ ,  $\mathcal{V}_T$  and  $\mathcal{V}_P$  respectively, and they alternate their moves, i.e.,  $\mathcal{E} \subset (\mathcal{V}_D \times \mathcal{V}_T) \cup (\mathcal{V}_T \times \mathcal{V}_P) \cup (\mathcal{V}_P \times \mathcal{V}_D)$ . We suppose that every edge  $(p, d) \in \mathcal{E}$  with  $p \in \mathcal{V}_P$  and  $d \in \mathcal{V}_D$  is equipped with a *multiplicity*  $m_{pd}$  which is a (positive) natural number. For simplicity of exposition, we shall define here the value of an entropy game using only pure policies. More precisely, a

(pure) policy  $\sigma$  of Despot is a map which assigns to every node  $d \in \mathcal{V}_D$  a node  $t$  such that  $(d, t) \in \mathcal{E}$ . Similarly, a policy  $\tau$  of Tribune is a map which assigns to every node  $t \in \mathcal{V}_T$  a node  $p \in \mathcal{V}_P$ . We denote by  $n$  the cardinality of  $\mathcal{V}_D$ . Such a pair of policies determine a  $n \times n$  matrix  $M^{\sigma, \tau}$ , such that  $M_{d, d'}^{\sigma, \tau} = m_{\tau(\sigma(d)), d'}$ . Given an initial state  $\bar{d} \in \mathcal{V}_D$ , we measure the “freedom” of Player People by the limit  $R(\sigma, \tau) := \lim_{k \rightarrow \infty} [(M^{\sigma, \tau})^k e]_{\bar{d}}^{1/k}$ . A pair of (pure) policies determine a subgraph  $\mathcal{G}^{\sigma, \tau}$ , obtained by keeping only the successor prescribed by  $\sigma$  for every node of  $\mathcal{V}_D$ , and similarly for  $\tau$  and  $\mathcal{V}_T$ . Then, the “freedom” of player People is precisely the geometric growth rate of the number of paths of length  $3k$  starting from node  $\bar{d}$ , counted with multiplicities, as  $k \rightarrow \infty$ . In general, the graph  $\mathcal{G}^{\sigma, \tau}$  may have several strongly connected components, and it is observed in [5] that  $R(\sigma, \tau)$  coincides with the maximal spectral radii of the diagonal blocks of the matrix  $M^{\sigma, \tau}$  corresponding to the strongly connected components to which the initial state  $\bar{d}$  has access in  $\mathcal{G}^{\sigma, \tau}$ . In an entropy game, Despot wishes to minimize the freedom of People, whereas Tribune (a reference to the magistrate of Roman republic) wishes to maximize it. It is shown in [1] that the entropy game has a value in the space of pure policies, meaning that there exists pure policies  $\sigma^*, \tau^*$ , such that  $R(\sigma^*, \tau) \leq R(\sigma^*, \tau^*) \leq R(\sigma, \tau^*)$  for all pure policies  $\sigma, \tau$ . (Actually, more general, history dependent, strategies are considered in [1], and it is shown there that pure policies are optimal).

The dynamic programming operator of an entropy game is the self-map  $F$  of  $\mathbb{R}_{>0}^n$  given by  $F_d(x) = \min_{t \in \mathcal{V}_T, (d, t) \in \mathcal{E}} \max_{p \in \mathcal{V}_P, (t, p) \in \mathcal{E}} \sum_{d' \in \mathcal{V}_D, (p, d') \in \mathcal{E}} m_{p, d'} x_{d'}$ , for  $d \in \mathcal{V}_D$ . Then, the operator  $T := \log \circ F \circ \exp$  is a Shapley operator. It is shown in [1] that the value of the entropy game with initial state  $\bar{d}$  is given by the limit  $\lim_{k \rightarrow \infty} [(F^k(e))_{\bar{d}}]^{1/k}$ .

#### 4 The unichain property

Recall that to every  $n \times n$  nonnegative matrix  $M$  is associated a digraph with set of nodes  $[n]$ , such that there is an arc from  $i$  to  $j$  if  $M_{ij} > 0$ . The matrix is *irreducible* if this digraph is strongly connected. It is *unichain* if this digraph has a unique final strongly connected component (a strongly components is *final* if any path starting from this component stays in this component). The property of unichainedness is sometimes referred to as *ergodicity* since a stochastic matrix is unichain iff it has only one invariant measure, or equivalently, if the only *harmonic vectors* (i.e. the solutions  $v$  of  $Mv = v$ ) are the constant vectors, see the discussion in Theorem 1.1 of [2], and the references therein.

Given a pair  $(\sigma, \tau) \in \Pi_P$  of pure policies, we define the stochastic matrix:  $P^{\sigma, \tau} = (P_{i, j}^{\sigma(i), \tau(i)})_{i, j \in [n]}$ .

► **Definition 7.** We say that a game is unichain (resp. irreducible) if for all pairs of pure policies  $\sigma, \tau$ , the matrix  $P^{\sigma, \tau}$  is unichain (resp. irreducible).

► **Definition 8.** We say that a subset  $S$  of the states is closed under the action of a matrix  $P^{\sigma, \tau}$  if, starting from a state  $s \in S$  and playing according to the policies  $\sigma$  and  $\tau$ , the next state is still in  $S$ .

► **Remark 9.** If  $S$  is a set closed under the action of an unichain matrix  $P^{\sigma, \tau}$ , then  $S$  contains the final class of this matrix.

► **Remark 10.** The final class does not have to be the same for all pairs  $\sigma, \tau$  of policies in our definition of unichain games.

The following theorem addresses the issue of the existence of a solution to the ergodic equation in the case of a unichain game.

► **Theorem 11.** *Let  $T$  be the Shapley operator of a unichain concurrent stochastic game. Then, there exists a vector  $v \in \mathbb{R}^n$  and  $\lambda \in \mathbb{R}$  such that  $T(v) = \lambda e + v$ . Moreover, there exists a pair of optimal (randomized) Markovian stationary strategies, obtained by selecting actions that achieve the minimum and maximum in the expression of  $[T(v)]_i$ , for each state  $i \in [n]$ .*

## 5 Relative value iteration

Relative value iteration was introduced in [38] to solve one player stochastic mean-payoff games (i.e., average cost Markov decision processes). The “vanilla” value iteration algorithm consists in computing the sequence  $x_{k+1} = T(x_k)$ , starting from  $x_0 = 0$ . Then,  $x_k$  yields the value vector of the game in horizon  $k$ , and so, we expect  $x_k$  to go to infinity as  $k \rightarrow \infty$ . The idea of *relative* value is to renormalize the sequence by additive constants. We state in Algorithm 1 a general version of relative value iteration, allowing for *approximate* dynamic programming oracles. This will allow us to obtain complexity results in the Turing model of computation, by computing a rational approximation of the value of the Shapley operator  $T(x)$  at a given rational vector  $x$  up to a given accuracy.

■ **Algorithm 1** Relative value iteration in approximate arithmetics.

- 
- 1: **input:** A final requested numerical precision  $\epsilon > 0$  and a parameter  $0 < \eta \leq \epsilon/3$ . An oracle  $\tilde{T}$  which provides an  $\eta$ -approximation in the sup-norm of a Shapley operator  $T$ .
  - 2:  $x := 0 \in \mathbb{R}^n$
  - 3: **repeat**
  - 4:    $x := \tilde{T}(x) - \mathbf{t}(\tilde{T}(x))e$
  - 5: **until**  $\|x - \tilde{T}(x)\|_H \leq \epsilon/3$
  - 6:  $\alpha := \mathbf{b}(\tilde{T}(x) - x); \beta := \mathbf{t}(\tilde{T}(x) - x)$
  - 7: **return**  $x, \alpha, \beta$    ▷ The lower and upper escape rates of  $T$  are included in the interval  $[\alpha - \epsilon/3, \beta + \epsilon/3]$ , which is of width at most  $\epsilon$
- 

► **Theorem 12.** *Suppose that  $T$  is a Shapley operator. Then,*

1. *When it terminates, Algorithm 1 returns a valid interval of width at most  $\epsilon$  containing the lower and upper escape rates of  $T$ .*
  2. *If there is an integer  $q$  and a scalar  $0 < \gamma < 1$  such that  $T^q$  is a  $\gamma$ -contraction in Hilbert’s seminorm, and if  $\eta$  is chosen small enough, in such a way that  $\eta(12 + 24q/(1 - \gamma)) \leq \epsilon$ , then Algorithm 1 terminates in at most  $q(\log \|T(0)\|_H + \log 6 + |\log \epsilon|/|\log \gamma|)$  iterations.*
- The proof exploits the nonexpansiveness of the operator  $T$  in Hilbert’s seminorm.

## 6 Krasnoselskii–Mann damping

We shall see that for turn-based or concurrent games, it is useful to replace the original Shapley operator by a Krasnoselskii–Mann damped version of this operator. This will allow the relative-value iteration algorithm to converge under milder conditions.

► **Definition 13.** *If  $T$  is a Shapley operator, and  $0 < \theta < 1$ , we define  $T_\theta = \theta I + (1 - \theta)T$  where  $I$  is the identity operator.*

We will call *Krasnoselskii–Mann operator* the  $T_\theta$  operator. It is easy to show that it is also a Shapley operator. The following observation relates the ergodic constant of a damped Shapley operator with the ergodic constant of the original Shapley operator.

## 10:10 Relative Krasnoselskii–Mann Iteration

► **Lemma 14.** *Let  $T$  be a Shapley operator,  $u \in \mathbb{R}^n$  and  $\lambda \in \mathbb{R}$ . Then,  $T(u) = \lambda e + u$  if and only if  $T_\theta(u) = (1 - \theta)\lambda e + u$ . In particular,  $\chi(T) = (1 - \theta)^{-1}\chi(T_\theta)$  holds as soon as the ergodic equation  $T(u) = \lambda e + u$  is solvable.*

**Proof.** The equivalence is straightforward, and  $\chi(T) = (1 - \theta)^{-1}\chi(T_\theta)$  follows from Obs. 4. ◀

We consider the iteration  $x_{k+1} = T_\theta(x_k) - \mathbf{t}(T_\theta(x_k))e$ , obtained by applying relative value iteration (as in Algorithm 1) to the Krasnoselskii–Mann operator  $T_\theta$ , with an arbitrary initial condition  $x_0 \in \mathbb{R}^n$ . Ishikawa showed that the ordinary Krasnoselskii–Mann iteration applied to a nonexpansive self-map of a finite dimensional normed space does converge, as soon as a fixed point exists [24]. This entails the following result.

► **Theorem 15** (Compare with [19]). *Let  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a Shapley operator, and  $0 < \theta < 1$ . Then, the sequence  $x_k$  obtained by applying relative value iteration to the Krasnoselskii–Mann operator  $T_\theta$  converges if and only if  $\exists u \in \mathbb{R}^n, \lambda \in \mathbb{R}, T(u) = \lambda e + u$ .*

A multiplicative variant of this result was proved in Theorem 11 of [19].

## 7 Contraction properties of unchain games under pure policies

We define the following parameter, representing the minimal value of a non-zero off-diagonal transition probability,  $p_{\min} = \min_{i,j \in [n], i \neq j, (a,b) \in A \times B} \{P_{i,j}^{a,b} : P_{i,j}^{a,b} > 0\}$ , and set  $\theta := p_{\min}/(1 + p_{\min})$ . For every pair of policies  $\sigma, \tau$  of the two players, we set  $Q^{\sigma,\tau} := \theta I + (1 - \theta)P^{\sigma,\tau}$ . For any sequence of pairs of pure policies  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$ , we define, for all  $i \in [n]$ ,  $S_i(\sigma_1, \tau_1, \dots, \sigma_k, \tau_k) := \{j \mid [Q^{\sigma_1, \tau_1} \dots Q^{\sigma_k, \tau_k}]_{ij} > 0\}$ .

► **Lemma 16.** *Suppose a concurrent game is unichain. Then, there is an integer  $k \leq n$  such that for all  $i_1, i_2 \in [n]$ , and for all sequences of pairs of pure policies  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$ ,  $S_{i_1}(\sigma_1, \tau_1, \dots, \sigma_k, \tau_k) \cap S_{i_2}(\sigma_1, \tau_1, \dots, \sigma_k, \tau_k) \neq \emptyset$ .*

We call the *unchain index* of the game, and denote by  $k_{\text{uni}}$  the smallest integer  $k$  satisfying the property of Lemma 16. Similarly, we call *irreducibility index* of an irreducible game, and denote by  $k_{\text{irr}}$ , the smallest integer  $k$  such that for every sequence of pure policies  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$ , the matrix  $Q^{\sigma_1, \tau_1} \dots Q^{\sigma_k, \tau_k}$  is positive. We have  $1 \leq k_{\text{uni}} \leq k_{\text{irr}}$ .

The following result will allow us to obtain a geometric contraction rate. The proofs of this theorem and of the next proposition show in particular that  $k_{\text{uni}} \leq n$  if the game is unichain and  $k_{\text{irr}} \leq n$  if the game is irreducible.

► **Theorem 17.** *Let us suppose that a concurrent game with  $n$  states is unichain, with unichain index  $k = k_{\text{uni}}$ . Then, for all sequences  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$  of pairs of pure policies of the two players,  $\|Q^{\sigma_1, \tau_1} \dots Q^{\sigma_k, \tau_k}\|_{\text{H}} \leq 1 - \theta^k$ .*

The following proposition improves the bound on the contraction rate provided by Theorem 17, in the special case of irreducible games.

► **Proposition 18.** *Let us suppose that a concurrent game with  $n$  states is irreducible, and let  $k = k_{\text{irr}}$  be the irreducibility index of the game. Then, for all sequences  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$  of pairs of pure policies of the two players,  $\|Q^{\sigma_1, \tau_1} \dots Q^{\sigma_k, \tau_k}\|_{\text{H}} \leq 1 - n\theta^k$ .*

## 8 Solving concurrent and turn-based games by relative Krasnoselskii–Mann iteration

We first establish a general bound for concurrent unichain games. Recall that  $\theta = p_{\min}/(1 + p_{\min})$ .

► **Theorem 19.** *Let  $T_\theta$  be the Krasnoselskii–Mann operator of a concurrent and unichain game. Then,  $T_\theta^{k_{\text{uni}}}$  is a contraction in Hilbert’s seminorm, with rate bounded by  $1 - \theta^{k_{\text{uni}}}$ . Moreover, if the game is irreducible,  $T_\theta^{k_{\text{irr}}}$  is a contraction in Hilbert’s seminorm, with rate bounded by  $1 - n\theta^{k_{\text{irr}}}$ .*

Combining this result with Theorem 12, we obtain the following result, in which we denote by  $\|r\|_\infty := \max_{i,a,b} |r_i^{ab}|$  the sup-norm of the payment function.

► **Corollary 20.** *Let  $T$  be the Shapley operator of a concurrent unichain game, and  $\epsilon \in (0, 1)$ . Algorithm 1, applied to the Krasnoselskii–Mann operator  $T_\theta$ , with the precision  $\eta$  prescribed in Theorem 12, provides an  $\epsilon$ -approximation of the value of the game in at most  $(|\log(\epsilon)| + \log 24 + \log \|r\|_\infty)k_{\text{uni}}\theta^{-k_{\text{uni}}}$  iterations.*

We now consider the special case of turn-based games. Then, the value is a rational number, and there are optimal pure policies. We now apply our approach to compute exactly the value and to find optimal pure policies.

► **Assumption 21.** *We now assume that the probabilities  $P_{i,j}^{a,b}$  are rational numbers with a common denominator denoted by  $M$ . We also assume that the payments  $r_i^{a,b}$  are integers.*

► **Lemma 22** (Coro. of [35]). *Let  $P$  be a  $n \times n$  unichain matrix whose entries are rational numbers with a common denominator  $M$ . Then, the entries of the unique invariant measure of  $P$  are rational numbers of denominator at most  $nM^{n-1}$ .*

When Algorithm 1 halts, returning a vector  $x \in \mathbb{R}^n$ , we select two pure policies  $\sigma^*$  and  $\tau^*$  that reach the minimum and maximum in the expression of  $T(x)$ , meaning that, for  $i \in [n]$ , we have:

$$T_i(x) = \max_{b \in B(i)} \left( r_i^{\sigma^*(i),b} + \sum_{j \in [n]} P_{ij}^{\sigma^*(i),b} x_j \right) = \min_{a \in A(i)} \left( r_i^{a,\tau^*(i)} + \sum_{j \in [n]} P_{ij}^{a,\tau^*(i)} x_j \right). \quad (4)$$

► **Theorem 23.** *Consider a unichain turn-based stochastic game satisfying Assumption 21. Let us choose  $\epsilon = (1 - \theta)(n^2 M^{2(n-1)})^{-1}$ , so that Algorithm 1 applied to  $T_\theta$  runs in at most*

$$(2 \log n + 2(n - 1) \log M + \log 24 + \log \|r\|_\infty) \theta^{-k_{\text{uni}}} k_{\text{uni}} \quad (5)$$

*iterations. Let  $x^*$  be the vector returned by the algorithm. Let us select pure policies  $\sigma^*$  and  $\tau^*$  reaching respectively the minimum and maximum in the expression of  $T(x^*)$ , as in (4). Then, these policies are optimal.*

## 9 Multiplicative Krasnoselskii–Mann Damping applied to Entropy Games

In the case of entropy games, the ergodic eigenproblem, for the operator  $F$  defined in Section 3.2, consists in finding  $u \in \mathbb{R}^n$  and  $\lambda \in \mathbb{R}$  such that  $\exp(\lambda) \exp(u) = F(u)$ . Equivalently,  $\lambda e + u = T(u)$  where  $T = \log \circ F \circ \exp$ . If this equation is solvable, then  $\exp(\lambda)$  is the value of the entropy game, for all initial states  $d \in \mathcal{V}_D$ . To solve this equation, we fix a positive number  $\vartheta > 0$ , and consider the following “multiplicative” variant of the Krasnoselskii–Mann operator:

## 10:12 Relative Krasnoselskii–Mann Iteration

$$[T_{m,\vartheta}(v)]_d = \log \min_{t \in \mathcal{V}_T, (d,t) \in \mathcal{E}} \max_{p \in \mathcal{V}_P, (t,p) \in \mathcal{E}} (\vartheta \exp(v_d) + \sum_{d' \in \mathcal{V}_D, (p,d') \in \mathcal{E}} m_{p,d'} \exp(v_{d'})) ,$$

recalling that  $m_{p,d'}$  denotes the multiplicity of the arc  $(p, d')$ . Unlike in the additive case, we do not perform a “convex combination” of the identity map and of the Shapley operator, but we only add the “diagonal term”  $\vartheta \exp(v_d)$ , where  $\vartheta$  can still be interpreted as a “damping intensity”, albeit in a multiplicative sense. If  $T(u) = \lambda e + u$ , then, one readily checks that  $T_{m,\vartheta}(u) = \mu e + u$ , where  $\mu = \log(\vartheta + \exp(\lambda))$ , and vice versa, so the non-linear eigenproblems for  $T$  and  $T_{m,\vartheta}$  are equivalent. As in the additive case, the damping intensity must be tuned to optimize the complexity bounds. We shall say that the multiplicity  $m_{p,d'}$  is *off-diagonal* if there is no path  $d' \rightarrow t \rightarrow p \rightarrow d'$  in the graph of the game. Equivalently, for any choices of policies  $\sigma, \tau$  of the two players, the entry  $m_{p,d'}$  does not appear on the diagonal of the matrix  $M^{\sigma,\tau}$ , defined in Section 3.2. Then, we denote by  $\underline{m}$  the minimum of off-diagonal multiplicities, observe that  $\underline{m}$  is precisely the minimum of all off-diagonal entries of the matrices  $M^{\sigma,\tau}$  associated to all pairs of policies. We set  $\vartheta := \underline{m}$ .

We shall say that an entropy game is *irreducible* if for every pair of policies  $\sigma, \tau$ , the matrix  $M^{\sigma,\tau}$  is irreducible. The *irreducibility index*  $k_{\text{irr}}$  of an irreducible entropy game is the smallest integer  $k$  such that for all policies  $\sigma_1, \tau_1, \dots, \sigma_k, \tau_k$ , the matrix  $M^{\sigma_1\tau_1} \dots M^{\sigma_k\tau_k}$  has positive entries. Arguing as in the case of stochastic concurrent games, we get that  $k_{\text{irr}} \leq n$  as soon as the game is irreducible. We define the *l-ambiguity* of the entropy game  $\mathcal{A}_l := \max_{d,d' \in \mathcal{V}_D} \max_{\sigma_1, \tau_1, \dots, \sigma_l, \tau_l} (M^{\sigma_1\tau_1} \dots M^{\sigma_l\tau_l})_{d,d'}$ . Observe that  $(M^{\sigma_1\tau_1} \dots M^{\sigma_l\tau_l})_{d,d'}$  is the number of paths from  $d$  to  $d'$  counted with multiplicities, in the finite horizon game induced by the policies  $\sigma_1, \tau_1, \dots, \sigma_l, \tau_l$  (this motivates the term “l-ambiguity”). If the game is irreducible, we define the *ambiguity* of the game  $\mathcal{A} := \max_{1 \leq l \leq k_{\text{irr}}} \mathcal{A}_l^{1/l}$ . We set  $W := \max_{(p,d) \in \mathcal{E} \cap (\mathcal{V}_P \times \mathcal{V}_D)} m_{p,d}$ , and observe that  $W \leq \mathcal{A} \leq n^{1-1/k_{\text{irr}}} W$ .

► **Theorem 24.** *Let  $T_{m,\vartheta}$  be the multiplicative Krasnoselskii–Mann operator of an irreducible entropy game. Then,  $T_{m,\vartheta}^{k_{\text{irr}}}$  is a contraction in Hilbert’s seminorm, with contraction rate bounded by  $\frac{\bar{\mathcal{M}}-1}{\bar{\mathcal{M}}+1}$ , where  $\bar{\mathcal{M}} := (1 + \mathcal{A}/\underline{m})^{k_{\text{irr}}}$ .*

We recall the following separation bound.

► **Theorem 25** (Coro. of [5]). *Suppose two pairs of (pure) policies yield distinct values in an entropy game with  $n$  Despot’s states. Then, these values differ at least by  $\nu_n^{-1}$  where*

$$\nu_n := 2^n (n+1)^{8n} n^{2n^2+n+1} e^{4n^2} \max(1, W/2)^{4n^2} .$$

Then, using Theorem 12, we deduce:

► **Theorem 26.** *Consider an irreducible entropy game, with irreducibility index  $k_{\text{irr}}$ . Let us choose  $\epsilon = (1 + (\underline{m} + W)\nu_n)^{-1}$ , so that Algorithm 1 applied to  $T_{m,\vartheta}$  runs in at most  $(\log(1 + (\underline{m} + W)\nu_n) + \log 6)k_{\text{irr}}\bar{\mathcal{M}}/2$  iterations. Moreover, let  $x^*$  be the vector returned by the algorithm. Let us select pure policies  $\sigma^*$  and  $\tau^*$  reaching respectively the minimum and maximum in the expression of  $T_{m,\vartheta}(x^*)$ . Then, these policies are optimal.*

## 10 Concluding Remarks

We have established parameterized complexity bounds for relative value iteration applied to several classes of stochastic games satisfying irreducibility conditions. These bounds rely on contraction properties in Hilbert’s seminorm. It would be interesting to see whether these contraction properties can also be exploited to derive complexity bounds for policy iteration, instead of value iteration.

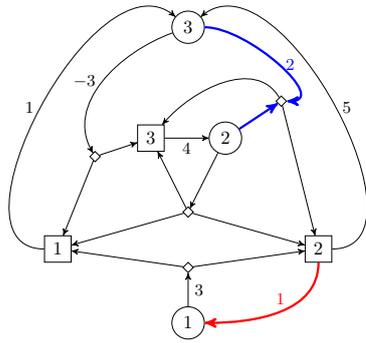
## References

- 1 M. Akian, S. Gaubert, J. Grand-Clément, and J. Guillaud. The operator approach to entropy games. *Theory of Computing Systems*, 63:1089–1130, 2019.
- 2 M. Akian, S. Gaubert, and A. Hochart. Ergodicity conditions for zero-sum games. *Discrete Contin. Dyn. Syst.*, 35(9):3901–3931, 2015.
- 3 M. Akian, A. Sulem, and M. I. Taksar. Dynamic optimization of long-term growth rate for a portfolio with transaction costs and logarithmic utility. *Mathematical Finance*, 11(2):153–188, April 2001.
- 4 Marianne Akian, Stéphane Gaubert, Ulysse Naepels, and Basile Terver. Solving irreducible stochastic mean-payoff games and entropy games by relative Krasnoselskii–Mann iteration, 2023. Extended version of the present article, arXiv:2305.02458.
- 5 X. Allamigeon, S. Gaubert, R. D. Katz, and M. Skomra. Universal Complexity Bounds Based on Value Iteration and Application to Entropy Games. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 110:1–110:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 6 V. Anantharam and V. S. Borkar. A variational formula for risk-sensitive reward. *SIAM J. Contro. Optim.*, 55(2):961–988, 2017.
- 7 D. Andersson and P. B. Miltersen. The complexity of solving stochastic games on graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Comput. Sci.*, pages 112–121. Springer, 2009.
- 8 E. Asarin, J. Cervelle, A. Degorre, C. Dima, F. Horn, and V. Kozyakin. Entropy games and matrix multiplication games. In *Proceedings of the 33rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 11:1–11:14, Wadern, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- 9 L. Attia and M. Oliu-Barton. A formula for the value of a stochastic game. *PNAS*, 52(116):26435–26443, 2019.
- 10 J. B. Baillon and R. E. Bruck. Optimal rates of asymptotic regularity for averaged nonexpansive mappings. In K. K. Tan, editor, *Proceedings of the Second International Conference on Fixed Point Theory and Applications*, pages 27–66. World Scientific Press, 1992.
- 11 T. Bewley and E. Kohlberg. The asymptotic theory of stochastic games. *Math. Oper. Res.*, 1(3):197–208, 1976.
- 12 E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. A potential reduction algorithm for two-person zero-sum mean payoff stochastic games. *Dynamic Games and Applications*, 8(1):22–41, July 2018.
- 13 K. Chatterjee and R. Ibsen-Jensen. The complexity of ergodic mean-payoff games. Extended version of a paper published in the proceedings of ICALP, 2014. arXiv:1404.5734.
- 14 A. Condon. The complexity of stochastic games. *Inform. and Comput.*, 96(2):203–224, 1992.
- 15 R. L. Dobrushin. Central limit theorem for nonstationary Markov chains. I. *Theory of Probability & Its Applications*, 1(1):65–80, January 1956.
- 16 K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. *Logical Methods in Computer Science*, 4(4), November 2008.
- 17 A Federgruen, P.J Schweitzer, and H.C Tijms. Contraction mappings underlying undiscounted Markov decision problems. *Journal of Mathematical Analysis and Applications*, 65(3):711–730, 1978.
- 18 S. Gaubert and J. Gunawardena. The Perron-Frobenius theorem for homogeneous, monotone functions. *Trans. of AMS*, 356(12):4931–4950, 2004.
- 19 S. Gaubert and N. Stott. A convergent hierarchy of non-linear eigenproblems to compute the joint spectral radius of nonnegative matrices. *Mathematical Control and Related Fields*, 10(3):573–590, 2020.

- 20 D. Gillette. *Stochastic games with zero stop probabilities*, volume III, chapter 9, pages 179–188. Princeton University Press, 1958.
- 21 K. Arnsfelt Hansen, M. Koucky, N. Lauritzen, P. Bro Miltersen, and E. P. Tsigaridas. Exact algorithms for solving stochastic games. In *STOC 2011*, 2011.
- 22 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Manag. Sci.*, 12(5):359–370, 1966.
- 23 R. A. Howard and J. E. Matheson. Risk-sensitive Markov decision processes. *Management Science*, 18(7):356–369, 1972.
- 24 S. Ishikawa. Fixed points and iteration of a nonexpansive mapping in a Banach space. *Proceedings of the American Mathematical Society*, 59(1):65–71, 1976.
- 25 M. A. Krasnosel’skiĭ. Two remarks on the method of successive approximations. *Uspekhi Matematicheskikh Nauk*, 10:123–127, 1955.
- 26 T. M. Liggett and S. A. Lippman. Stochastic games with perfect information and time average payoff. *SIAM Rev.*, 11:604–607, 1969.
- 27 W. R. Mann. Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4:506–510, 1953.
- 28 J.-F. Mertens and A. Neyman. Stochastic games. *Internat. J. Game Theory*, 10(2):53–66, 1981.
- 29 J.-F. Mertens, S. Sorin, and S. Zamir. *Repeated games*, volume 55 of *Econom. Soc. Monogr.* Cambridge University Press, Cambridge, 2015.
- 30 H.D. Mills. Marginal values of matrix games and linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, volume 38 of *Annals of Mathematics Studies*, pages 183–194. Princeton University Press, 1956.
- 31 D. Rosenberg and S. Sorin. An operator approach to zero-sum repeated games. *Israel J. Math.*, 121(1):221–246, 2001.
- 32 U. G. Rothblum. Multiplicative Markov decision chains. *Mathematics of Operations Research*, 9(1):6–24, 1984.
- 33 U. G. Rothblum and P. Whittle. Growth optimality for branching Markov decision chains. *Mathematics of Operations Research*, 7(4):582–601, 1982.
- 34 L. S. Shapley. Stochastic games. *Proc. Natl. Acad. Sci. USA*, 39(10):1095–1100, 1953.
- 35 M. Skomra. Optimal bounds for bit-sizes of stationary distributions in finite Markov chains. Preprint arxiv:2109.04976, 2021.
- 36 K. Sladký. *On dynamic programming recursions for multiplicative Markov decision chains*, pages 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976.
- 37 G. Vigeral. A zero-sum stochastic game with compact action sets and no asymptotic value. *Dynamic Games and Applications*, 3(2):172–186, January 2013.
- 38 D.J White. Dynamic programming, Markov chains, and the method of successive approximations. *Journal of Mathematical Analysis and Applications*, 6(3):373–376, 1963.
- 39 W. H. M. Zijm. Asymptotic expansions for dynamic programming recursions with general nonnegative matrices. *J. Optim. Theory Appl.*, 54(1):157–191, 1987.
- 40 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoret. Comput. Sci.*, 158(1–2):343–359, 1996.

## **A** Example of turn-based stochastic mean-payoff game

A turn-based stochastic mean-payoff game is represented below. Min states are represented by squares; Max states are represented by circles; Nature states are represented by small diamonds. The payments made by Min to Max are shown on the arcs. For every Nature state, the next state is chosen with the uniform distribution among the successors. The associated Shapley operator is the map  $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$  shown at right.



$$T_1(x) = 1 + \max\left(2 + \frac{x_2 + x_3}{2}, -3 + \frac{x_1 + x_3}{2}\right)$$

$$T_2(x) = \min\left(5 + \max\left(2 + \frac{x_2 + x_3}{2}, -3 + \frac{x_1 + x_3}{2}\right), 1 + 3 + \frac{x_1 + x_2}{2}\right)$$

$$T_3(x) = 4 + \max\left(\frac{x_2 + x_3}{2}, \frac{x_1 + x_2 + x_3}{3}\right)$$

The unichain index defined in Section 7 is  $k_{\text{uni}} = 1$ . Indeed, for all pairs of policies  $(\sigma_1, \tau_1)$ , we have  $S_1(\sigma_1, \tau_1) \supset \{1, 3\}$ ,  $S_3(\sigma_1, \tau_1) \supset \{2, 3\}$ , and  $S_2(\sigma_1, \tau_1) \supset \{2, 3\}$  if  $\sigma_1$  sends Min state 2 to Max state 3, and  $S_2(\sigma_1, \tau_1) = \{1, 2\}$  if  $\sigma_1$  sends Min state 2 to Max state 1. In all cases, we have  $S_i(\sigma_1, \tau_1) \cap S_j(\sigma_1, \tau_1) \neq \emptyset$  for  $i \neq j$ . We have  $p_{\min} = 1/3$ , and  $\theta = p_{\min}/(1 + p_{\min}) = 1/4$ . It follows from Theorem 19 that the damped Shapley operator  $T_\theta$  is a contraction of rate  $3/4$ . We know from Theorem 11 that the ergodic eigenproblem is solvable. By applying Algorithm 1, we find  $T(u) = \lambda e + u$  with  $(-1, -0.5, 0)$  and  $\lambda = 3.75$ . An approximation of  $u$  of precision  $< 10^{-8}$  in the sup norm is reached after only 15 iterations, to be compared with the precision of order  $(3/4)^{15} \simeq 10^{-2}$  given by the theoretical upper bound, for the same number of iterations. Thus, the convergence may be faster in practice than the one shown in Corollary 20. We deduce from  $T(u) = \lambda e + u$  that the value of the mean-payoff game is 3.75 regardless of the initial state. Optimal policies  $\sigma$  and  $\tau$  of both players are obtained by selecting the actions that achieve the minimum or the maximum in the expression of  $T(u)$ . The non-trivial actions of these optimal policies are as follows: from Min state 2 (square at bottom right), go to Max state 3 (circle at the top level), from Max state 2 (circle at the middle level), and also from Max state 3, got to the top right state (diamond) of Nature. These policies are shown on the figure above (red: policy of Min; blue: policy of Max). The stochastic matrix  $P^{\sigma, \tau}$  and payment vector  $r^{\sigma, \tau}$  associated to these policies are given by

$$r^{\sigma, \tau} = \begin{pmatrix} 3 \\ 4 \\ 4 \end{pmatrix}, \quad P^{\sigma, \tau} = \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \end{pmatrix}.$$

The unique invariant measure of the matrix  $P^{\sigma, \tau}$  is  $\pi = (1/4, 1/2, 1/4)$ , and we have  $\pi r^{\sigma, \tau} = 15/4 = 3.75$ , consistently with the value of the mean-payoff already found.



# The Geometry of Reachability in Continuous Vector Addition Systems with States

Shaull Almagor ✉ 

Technion – Israel Institute of Technology, Haifa, Israel

Arka Ghosh ✉

University of Warsaw, Poland

Tim Leys ✉ 

University of Antwerp – Flanders Make, Belgium

Guillermo A. Pérez ✉ 

University of Antwerp – Flanders Make, Belgium

---

## Abstract

We study the geometry of reachability sets of continuous vector addition systems with states (VASS). In particular we establish that they are “almost” Minkowski sums of convex cones and zonotopes generated by the vectors labelling the transitions of the VASS. We use the latter to prove that short so-called linear path schemes suffice as witnesses of reachability in continuous VASS. Then, we give new polynomial-time algorithms for the reachability problem for linear path schemes. Finally, we also establish that enriching the model with zero tests makes the reachability problem intractable already for linear path schemes of dimension two.

**2012 ACM Subject Classification** Theory of computation → Concurrency; Theory of computation → Logic and verification

**Keywords and phrases** Vector addition system with states, reachability, continuous approximation

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.11

**Related Version** *Full Version*: <https://arxiv.org/abs/2210.00785>

**Funding** *Shaull Almagor*: supported by the ISRAEL SCIENCE FOUNDATION (grant No. 989/22). *Arka Ghosh*: partially supported by the NCN grant 2019/35/B/ST6/02322.

**Acknowledgements** We thank anonymous reviewers for their careful reading of a previous version of this work and for their suggestions that greatly improved the presentation of this paper.

## 1 Introduction

Vector Addition Systems with States (VASS, for short) are rich mathematical models for the description of distributed systems, as well as chemical and biological processes, and more [19]. A VASS essentially consists of a finite-state machine whose transitions are labelled with integer vectors. Besides the current state, a configuration of the VASS also comprises the current values of a set of counters. When a transition is taken, the state of the machine changes and the values of the counters are updated by adding to them the vector that labels the transition. VASS arise naturally as an arguably-cleaner model than Petri nets, due to their reachability problem being polytime-interreducible with that of Petri nets.

While VASS are a very expressive model of concurrency that admits algorithmic analysis, the complexity of several associated decision problems is prohibitively high. For instance, the *reachability problem*, which asks “is a given target configuration reachable from a given initial configuration?”, was recently proved to be Ackermann-complete [6, 15, 14].

*Continuous VASS* were introduced by Blondin and Haase [2] as an alternative to *continuous Petri nets* [7] which trade off the ability to encode discrete information in favor of computational and practical benefits. Their only difference compared to VASS concerns how the counters are updated: In continuous VASS, when a transition is taken, the machine is



© Shaull Almagor, Arka Ghosh, Tim Leys, and Guillermo A. Pérez;  
licensed under Creative Commons License CC-BY 4.0

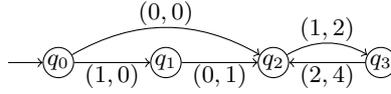
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 11; pp. 11:1–11:13

Leibniz International Proceedings in Informatics



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



■ **Figure 1** From  $q_0$ , with initial counter values  $\mathbf{0}$ , the state  $q_2$  can be reached with counter values  $\{(3i + a, 6i + b) \mid (a, b) \in \{(0, 0), (1, 1)\}, i \in \mathbb{N}\}$ ; with continuous semantics, it can be reached with counter values  $\mathbf{x} + \mathbf{y}$  where  $\mathbf{x} \in \{(0, 0)\} \cup \{(a, b) \mid 0 < a, b \leq 1\}$  and  $\mathbf{y}$  lies on the ray  $\{(i, 2i) \mid i \in \mathbb{N}\}$ .

allowed to scale the update vector by some scalar  $0 < \alpha \leq 1$  before adding it to the current counter values (see Figure 1). In contrast to the situation with “discrete” VASS, the computational complexity of the reachability problem for continuous VASS is relatively low. Namely, in [2] the reachability problem for continuous VASS is shown to be **NP**-complete while the complexity of the same problem for discrete VASS is Ackermann-complete [6, 15, 14].

Despite the relatively low computational complexity, **NP** is not universally considered as tractable. The only subcase previously known to be in **P** was that of cyclic reachability when counters are allowed to hold negative values. It is also worth noting that the aforementioned **NP** upper bound is obtained by encoding the reachability problem into the existential fragment of the first-order theory of the reals with addition and order. It is natural to ask whether more efficient algorithms or encodings into “simpler” logics exist, e.g. linear programming, even if only for particular subcases.

**Fixed-dimension VASS.** The relatively new Ackermann lower bound for VASS reachability has renewed interest in what could be named the *Bordeaux-Warsaw program*: the study of the computational complexity of the reachability problem for low-dimensional VASS and extensions thereof (see, e.g., [4, 3, 5, 8]). In such cases, there may be efficient algorithms for the problem and, to quote Czerwiński and Orlikowski [6], “it is easier to [design] sophisticated techniques working in a simpler setting [that might] result in finding new techniques useful in much broader generality.” For dimensions 1 and 2 (and counter updates encoded in binary) the problem is **NP**-complete [12] and **PSPACE**-complete [1], respectively.

An important structural restriction on VASS which is often used as an intermediate step in establishing upper bounds is that of *flatness*, i.e. disallowing nested cycles. In fact, the upper bounds for dimensions 1 and 2 mentioned above can be seen as a consequence of such VASS being effectively flattable [16]. A further restriction consists of asking that the set of all runs of the VASS can be represented by a single regular expression  $\pi_0 \chi_1^+ \dots \pi_{n-1} \chi_n^+ \pi_n$  over the transitions. Such VASS are called *linear path schemes* (LPS, for short). Linear path schemes played an essential role in [1], where it is shown that for any path that witnesses reachability, there exists a linear path scheme that also witnesses reachability.

**VASS variants.** In this work we study continuous VASS. For complexity matters, we assume all counter updates are encoded in binary. As decision problems, we focus on reachability (via a path that might make the counters negative); nonnegative reachability, i.e. reachability via a path that keeps the counters nonnegative at all times; and zero-test reachability, corresponding to reachability with the added constraint that some states can only be visited with value zero for a designated counter. We summarize known and new results in Table 1. Below, we give a textual account of the complexity bounds from the table.

**(Discrete) Reachability.** The **NP**-hardness bound for LPS can be shown using a simple reduction from the SUBSETSUM problem with multiplicities, i.e. summands can be added more than once. The latter is known to be **NP**-complete, see e.g. [9, Proposition 4.1.1]. The upper bound for the general case is folklore and is proven in [11] even with *resets*.

**Continuous reachability.** The **NP**-hardness bound for flat VASS is stated in [2, Lemma 4.13(a)] for nonnegative reachability but the reduction establishes it for reachability as well. The upper bound for the general case follows from [2, Corollary 4.10]. Membership in **P** for LPS can be derived from [2, Theorem 4.15] which states that continuous cyclic reachability is in **P**. In this work, we give an alternative algorithm for continuous cyclic reachability and present a full decision procedure for continuous reachability for LPS.

**Nonnegative reachability.** For fixed dimension  $d$ , only an  $F_{d+O(1)}$  upper bound is known [15]. **NP**-hardness for LPS follows from the same proof as for reachability since the construction has no negative updates. Finally, the **NP** upper bound for flat VASS is folklore: (nonnegative) reachability in flat VASS can be encoded into existential Presburger Arithmetic (PA), a theory whose decidability is **NP**-complete (see, e.g., [10]).

**Continuous nonnegative reachability.** The **NP** upper and lower bounds for the general and flat cases follow from (the proofs of) the same results in the continuous reachability case. For the **P** upper bound, however, one cannot rely on [2, Theorem 4.15]. In fact, cyclic reachability (for general dimensions) is **NP**-hard in the continuous nonnegative case [2, Lemma 4.13(b)]. This is, thus, the first novel complexity bound we establish.

**Zero-test reachability.** The **NP**-hardness bound for LPS is a consequence of reachability being a subcase of zero-test reachability. The matching upper bound for flat VASS is an extension of the classical encoding into PA which accounts for linear constraints imposed by the zero tests on cycles. Finally, the general model is also known as Minsky machines and its reachability problem was proven undecidable by Minsky himself [17].

**Continuous zero-test reachability.** The **NP**-hardness for flat VASS is a consequence of reachability being a subcase of zero-test reachability. For LPS, the lower bound is novel and points to continuous zero-test reachability not being a suitable approximation of the discrete case. The general case is undecidable in dimension 4 or higher [2, Theorem 4.17].

**Our contributions.** Our main contribution is a geometrical understanding of the reachability sets of continuous VASS (see Theorem 1, Theorem 5, and Theorem 7). The latter allows us (1) to prove that short LPS suffice as witnesses of (nonnegative) reachability (see Theorem 11 and Theorem 18), and (2) to give new algorithms for the reachability problem for LPS (see Theorem 13 and Theorem 19) via encodings of their reachability sets into tractable theories. Namely, we stay within linear programming solutions to enable efficient implementation of our algorithms. Finally, we establish that zero-test reachability for LPS is **NP**-hard even in dimension 2 (Theorem 21).

■ **Table 1** Summary of computational complexity results for the reachability problem for VASS of fixed dimension. We write lower bounds for simpler cases and upper bounds for more general ones. New results are shown in green (upper) and red (lower bounds).

Problem	Discrete			Continuous		
	General	Flat	LPS	General	Flat	LPS
Reachability	in <b>NP</b>	=	<b>NP</b> -hard	in <b>NP</b>	<b>NP</b> -hard	in <b>P</b>
Nonneg. reach.	in Ackermann	in <b>NP</b>	<b>NP</b> -hard	in <b>NP</b>	<b>NP</b> -hard	in <b>P</b>
Zero-test reach.	Undecidable	in <b>NP</b>	<b>NP</b> -hard	Undecidable	<b>NP</b> -hard	<b>NP</b> -hard

## 2 Preliminaries

In this work,  $\mathbb{Q}_{>0}$  denotes the set of all strictly positive rational numbers; and  $\mathbb{Q}_{\geq 0}$ , all nonnegative ones – including 0. Similarly,  $\mathbb{N}$ , i.e. the set of all natural numbers, includes 0, however  $\mathbb{N}_{>0}$  does not.

Let  $d$  be a positive integer. For any  $\mathbf{x} \in \mathbb{Q}^d$ , and  $r \in \mathbb{R}$ , we define the *open ball of radius  $r$  around  $\mathbf{x}$*  as usual:  $B_r(\mathbf{x}) = \{\mathbf{y} \in \mathbb{Q}^d : \|\mathbf{x} - \mathbf{y}\|_2 < r\}$ . Let  $X \subseteq \mathbb{Q}^d$ . Then, the *interior* of  $X$  is  $\text{int}(X) = \{\mathbf{x} \in X \mid \exists \varepsilon > 0, B_\varepsilon(\mathbf{x}) \subseteq X\}$ ; the *closure* of  $X$  is  $\text{cl}(X) = \{\mathbf{x} \in \mathbb{Q}^d \mid \forall \varepsilon > 0, B_\varepsilon(\mathbf{x}) \cap X \neq \emptyset\}$ ; and the *boundary* of  $X$  is  $\text{bd}(X) = \text{cl}(X) \setminus \text{int}(X)$ . Finally, we define the *relative interior* of  $X$ ,  $\text{relint}(X)$ , as its interior with respect to its embedding into its own affine hull as follows:  $\text{relint}(X) = \{\mathbf{x} \in X \mid \exists \varepsilon > 0, B_\varepsilon(\mathbf{x}) \cap \text{aff}(X) \subseteq X\}$ .

Let  $G \subseteq \mathbb{Q}^d$  be a set of (generating) vectors. We write  $\text{cone}(G)$  to denote the (*rational convex*) *cone*  $\{\sum_{i=0}^k a_i \mathbf{g}_i \mid k \in \mathbb{N}, \mathbf{g}_i \in G, a_i \in \mathbb{Q}_{\geq 0}\}$ . The (*linear*) *span* of  $G$  is defined as follows:  $\text{span}(G) = \{\sum_{i=0}^k a_i \mathbf{g}_i \mid k \in \mathbb{N}, \mathbf{g}_i \in G, a_i \in \mathbb{Q}\}$ . Finally, the *affine hull*  $\text{aff}(G)$  of  $G$  is the set  $\{\sum_{i=0}^k a_i \mathbf{g}_i \mid k \in \mathbb{N}, \mathbf{g}_i \in G, a_i \in \mathbb{Q}, \sum_{i=0}^k a_i = 1\}$ . (In particular, note that if  $\mathbf{0} \in G$  then  $\text{aff}(G) = \text{span}(G) = \text{span}(H)$ , for some  $H \subseteq G$  with cardinality at most  $d$ .)

### 2.1 Continuous VASS

Let  $d$  be a positive integer. A *continuous VASS*  $\mathcal{V}$  of dimension  $d$  is a tuple  $(Q, T, \ell)$  where  $Q$  is a finite set of states,  $T \subseteq Q \times Q$  is a finite set of transitions, and  $\ell : T \rightarrow \mathbb{Q}^d$  assigns an *update label* to every transition.

**Paths and runs.** A *configuration*  $c \in Q \times \mathbb{Q}^d$  is a tuple consisting of a state and the concrete values of the  $d$  *counters* of the VASS. We denote the configuration  $(p, \mathbf{x})$  by  $p(\mathbf{x})$ .

A *path*  $\pi$  is a sequence  $(p_1, p_2)(p_2, p_3) \dots (p_{n-1}, p_n) \in T^*$  of transitions. We write  $|\pi|$  to denote the length of the path, i.e.  $|\pi| = n - 1$ . A *run*  $\rho$  is a sequence  $q_1(\mathbf{x}_1)q_2(\mathbf{x}_2) \dots q_n(\mathbf{x}_n)$  of configurations such that for all  $1 \leq i < n$  we have:  $(q_i, q_{i+1}) \in T$  and  $\mathbf{x}_i + \alpha_i \cdot \ell(q_i, q_{i+1}) = \mathbf{x}_{i+1}$  for some  $\alpha_i \in \mathbb{Q}$  with  $0 < \alpha_i \leq 1$ . Often, we refer to the  $\alpha_i$  as the *coefficients of the run*. We say  $\rho$  *induces* the path  $(q_1, q_2) \dots (q_{n-1}, q_n)$ . Conversely, we sometimes say a run is *lifted* from a path. For instance,  $\pi$  can be lifted to a run  $p_1(\mathbf{y}_1) \dots p_n(\mathbf{y}_n)$  by fixing  $p_1(\mathbf{y}_1)$  as initial configuration and by choosing adequate coefficients  $\alpha_i$  for all transitions.

As a more concrete example, consider the path  $(q_0, q_1), (q_1, q_2), (q_2, q_3)$  in Figure 1, whose transitions are labelled by  $(1, 0)$  and  $(0, 1)$ . Starting from the configuration  $(0, 0)$  and using the coefficients  $\alpha_1 = 0.3$  and  $\alpha_2 = 0.5$  this path lifts to the run  $q_0(0, 0)q_1(0.3, 0)q_2(0.3, 0.5)$ .

We consider continuous VASS in a setting where only nonnegative counter values are allowed, denoted  $\mathbb{Q}_{\geq 0}$ VASS; and one which allows negative counters, denoted  $\mathbb{Q}$ VASS.

**Reachability.** Let  $p(\mathbf{x})$  and  $q(\mathbf{y})$  be two configurations. We say  $q(\mathbf{y})$  is *reachable* from  $p(\mathbf{x})$ , denoted  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$ , if there exists a run whose first and last configurations are  $p(\mathbf{x})$  and  $q(\mathbf{y})$  respectively. For a path  $\pi$ , we write  $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$  if, additionally, such a run exists which can be lifted from  $\pi$ . Given a configuration  $p_1(\mathbf{x})$  and a state  $q$ , we define the *reachability set* of a path  $\pi = (p_1, p_2) \dots (p_{n-1}, p_n)$  or a set  $P$  of paths below.

$$\text{Reach}^{\mathbf{x}}(\pi) = \{\mathbf{y} \in \mathbb{Q}^d \mid p_1(\mathbf{x}) \xrightarrow{\pi} p_n(\mathbf{y})\} \quad \text{Reach}^{\mathbf{x}}(P) = \bigcup_{\pi \in P} \text{Reach}^{\mathbf{x}}(\pi)$$

If  $\mathbf{x} = \mathbf{0}$  then we write simply  $\text{Reach}(\pi)$  and  $\text{Reach}(P)$ .

### 3 The Geometry of $\mathbb{Q}$ VASS Reachability Sets

In this section we discuss the geometry of the reachability sets in continuous VASS of dimension  $d$ . We first discuss paths and cycles separately. Then, we show that for solving the reachability problem, we only need to take short *linear path schemes* into consideration.

#### 3.1 The geometry of reachability sets for cycles

For this section, we fix a cycle  $\chi = (p_1, p_2) \dots (p_m, p_{m+1})$ , with  $p_1 = p_{m+1}$ . We study the geometry of the set  $\text{Reach}(\chi^+)$ , where  $\chi^+$  stands for  $\{\chi^k \mid k \in \mathbb{N}_{>0}\}$ .

Intuitively, following  $\chi$  allows us to add a scaled version of each transition vector along  $\chi$  arbitrarily many times, with the proviso that the scaling is strictly positive (the restriction to scale up to 1 disappears since we can repeatedly take the cycle). Thus, we can intuitively reach the interior of a cone, i.e. a positive linear combination of the vectors along  $\chi$ . For example, a cycle with vectors  $(1, 0)$  and  $(0, 1)$  will allow us to reach  $\{(x, y) \mid x > 0, y > 0\}$ . However, this intuition needs to be formalized carefully to account for linear dependencies between the vectors. This may render the cone not full-dimensional, i.e. its linear span may be a strict subspace of the vector space it is in. That would mean that the interior of the cone is empty. However, in such cases, the reachability set still is a “flattened” version of the interior, namely the relative interior of the cone.

##### 3.1.1 From cycles to cones

We formalize our intuition by proving that  $\text{Reach}(\chi^+)$  is the relative interior of the cone generated by  $G(\chi) = \{\ell(p_i, p_{i+1}) \mid 1 \leq i \leq m\}$ .

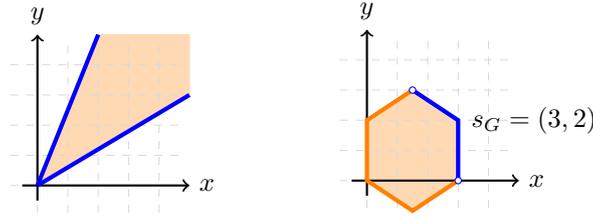
Indeed, all points  $\mathbf{x} \in \text{Reach}(\chi^+)$  can be obtained as positive linear combinations of generators. To any such  $\mathbf{x}$ , we can add or subtract any generating vector and stay within  $\text{cone}(G(\chi))$ , as long as it is sufficiently scaled down. Conversely, if one can add and subtract suitably scaled versions of all generating vectors to a point  $\mathbf{x} \in \text{cone}(G(\chi))$ , and remain within  $\text{cone}(G(\chi))$ , then it must be in the (relative) interior of  $\text{cone}(G(\chi))$ .

► **Theorem 1.** *Let  $G(\chi)$  be as defined above. Then,  $\text{Reach}(\chi^+) = \text{relint}(\text{cone}(G(\chi)))$ .*

Note that we consider the set  $G(\chi)$  of all labels of transitions from  $\chi$ , ignoring the fact that multiple transitions can have the same label. This is justified by the following lemma for the cycle  $\chi$  with generator  $G(\chi)$ .

► **Lemma 2.** *We have that  $\text{relint}(\text{cone}(\{\lambda_1 \ell(p_i, p_{i+1}) \mid 1 \leq i \leq m, \lambda_i \in \mathbb{Q}_{>0}\}))$  is equal to  $\{\sum_{i=1}^m a_i \mathbf{g}_i \mid a_i \in \mathbb{Q}_{>0}\}$ .*

**A concrete case: dimension 2.** For intuition, we state a consequence of Theorem 1 in dimension 2. For  $d = 2$ , a cone  $C$  is *trivial* if there exists  $\mathbf{v} \in \mathbb{Q}^d$  such that  $C$  is a subset of the line  $\{r \cdot \mathbf{v} \mid r \in \mathbb{Q}\}$ , and otherwise it is full-dimensional. For a trivial cone, its relative interior is either the entire cone (if the cone is the entire line), or the cone without  $\mathbf{0}$ , if the cone is a ray. It is easy to see that for a cycle  $\chi$  whose vector labellings  $G(\chi)$  are co-linear to  $\mathbf{v}$ , the reachability set of  $\chi^+$  in the continuous semantics is  $\text{cone}(G(\chi))$ , excluding any end-points (since the only possible end-point is  $\mathbf{0}$  if  $\text{cone}(G(\chi))$  is a ray). For full-dimensional cones, we can take any positive combination of the generators, but since no element of the generators can be taken zero times, the reachability set excludes the boundary. See Figure 2 (left) for a visualization. In the following statement we write  $\text{int}(G(\chi))$  and  $\text{bd}(G(\chi))$  for  $\text{int}(\text{cone}(G(\chi)))$  and  $\text{bd}(\text{cone}(G(\chi)))$  respectively.



■ **Figure 2** On the left: a cone with its boundary in blue; on the right: a zonotope with  $G = \{(1.5, 1), (1.5, -1), (0, 2)\}$ , where  $\text{adj}(G)$  is drawn in blue.

- **Corollary 3.** *Let  $G(\chi)$  be as above. In dimension  $d = 2$ , one of the following holds.*
- *Either  $\text{cone}(G(\chi))$  is trivial and  $\text{Reach}(\chi^+)$  is  $\text{cone}(G(\chi))$ , excluding any end-points;*
  - *or  $\text{cone}(G(\chi))$  is nontrivial and  $\text{Reach}(\chi^+) = \text{int}(G(\chi))$ .*

### 3.2 The geometry of reachability sets for paths

For this section, we fix a path  $\pi = (p_1, p_2) \dots (p_m, p_{m+1})$ . Similarly to the case of cycles, we establish a connection between  $\text{Reach}(\pi)$  and a type of polytope known as a zonotope.

Intuitively, since we now have a path that is taken once (rather than a cycle), the restriction that the scaling is at most 1 comes into play, and limits us. Moreover, multiplicities of linearly dependent vectors along the path also matter.

**Zonotopes.** Let  $G = \{\mathbf{g}_1, \dots, \mathbf{g}_k\} \subseteq \mathbb{Q}^d$  be a finite set of (generating) vectors. We write  $\text{zono}(G)$  to denote the *zonotope*<sup>1</sup>  $\{\sum_{j=1}^k a_j \mathbf{g}_j \mid a_j \in \mathbb{Q}, 0 \leq a_j \leq 1\}$ .

Following our interior-based approach for cycles, we study the reachable part of the boundary of a zonotope. We define  $s_G$  as the sum of all vectors in  $G$ , or  $s_G = \sum_{\mathbf{g} \in G} \mathbf{g}$ . Additionally, we define the set  $\text{adj}(G)$  of *faces* of  $\text{zono}(G)$  that are *adjacent* to  $s_G$  below. A face of  $\text{zono}(G)$  is any nonempty intersection of  $\text{zono}(G)$  with a half-space  $H$  such that none of the interior points of  $\text{zono}(G)$  lies on the boundary of  $H$ .

► **Definition 4 (Adjacent Sets).** *We define  $\text{adj}(G)$  as the union of  $\{s_G\}$  and all  $\mathbf{x} \in \mathbb{Q}^d$  on the relative interior of a face of  $\text{zono}(G)$  that contains  $s_G$ . (See Figure 2 for intuition.)*

Observe that  $\text{adj}(G) = \emptyset$  whenever  $s_G \in \text{int}(\text{zono}(G))$ .

#### 3.2.1 From paths to zonotopes

We show  $\text{Reach}(\pi)$  has a close relation with a zonotope whose generator is derived from the *multiset*  $M = [\ell(p_i, p_{i+1}) \mid 1 \leq i \leq m]$ . Intuitively, we obtain from  $M$  a generator  $G(\pi)$  by summing together co-linear vectors that are in the same “orientation”. For example, the vectors  $(1, 0)$  and  $(2, 0)$  along a single path have the same effect as  $(3, 0)$ , but  $(1, 0)$  and  $(-1, 0)$  have distinct effects. Technically, this is captured by grouping together vectors that are in the cones of each other. More formally, choose  $M' \subseteq M \setminus \{\mathbf{0}\}$  such that for all  $\mathbf{u} \in M \setminus \{\mathbf{0}\}$  there is a unique vector  $\mathbf{u}' \in M'$  such that  $\mathbf{u} \in \text{cone}(\mathbf{u}')$ . Then, define  $G(\pi)$  as follows:  $G(\pi) = \left\{ \sum_{\mathbf{u} \in M \cap \text{cone}(\mathbf{u}')} \mathbf{u} \mid \mathbf{u}' \in M' \right\}$ .

<sup>1</sup> Zonotope is the standard term, but since we do not use any of its special properties, the reader may view this as a standard polytope.

We show that the reachability set of  $\pi$  can be computed by taking  $\text{zono}(G(\pi))$ , and removing its boundary except for faces adjacent to  $s_G$ . The intuition behind this is similar to that of cycles: one can add strictly positive-scaled versions of the generating vectors, and therefore boundary elements that are obtained using 0-scales are unreachable. However, in zonotopes there are additional boundary faces that are obtained by capping the scale at 1 on some elements, and these are the faces adjacent to  $s_G$  (with  $s_G$  itself being the reachable vector where all elements are scaled to 1).

► **Theorem 5.** *Let  $G(\pi)$  be as above. Then,  $\text{Reach}(\pi) = \text{relint}(\text{cone}(G)) \cup \text{adj}(G(\pi))$ .*

As with cycles, compared to the multiset of all path labels, we restrict our attention to a simpler set  $G$  of vectors. The following result connecting them is almost immediate.

► **Lemma 6.** *Let  $\mathbf{x} \in \mathbb{Q}^d$ . Then, there exists  $(a_1, \dots, a_n) \in \mathbb{Q}^n$  such that  $\mathbf{x} = \sum_{i=1}^n a_i \mathbf{g}_i$  and  $0 < a_i \leq 1$ , for all  $1 \leq i \leq n$ , if and only if  $\mathbf{x} \in \text{Reach}(\pi)$ .*

### 3.3 The geometry of reachability sets for linear path schemes

A linear path scheme (LPS, for short) is a regular expression  $\pi_0 \chi_1^+ \pi_1 \dots \chi_n^+ \pi_n$  over the transitions. Importantly, the  $\pi_i$  are (possibly empty) paths; the  $\chi_i$  are cycles; and  $\pi_0 \chi_1 \pi_1 \dots \chi_n \pi_n$  is a valid path. Each LPS determines an infinite set  $\{\pi_0 \chi_1^{k_1} \dots \chi_n^{k_n} \pi_n \mid k_1, \dots, k_n \in \mathbb{N}_{>0}\}$  of paths that follows each of the paths exactly once and each of the cycles an arbitrary number of times. For this section, we fix a linear path scheme  $\sigma = \pi_0 \chi_1^+ \pi_1 \dots \chi_n^+ \pi_n$ .

#### 3.3.1 From LPS to cones and a zonotope

From previous developments in this work, the reader might already believe that the reachability set of an LPS can be shown to be the Minkowski sum of suitable subsets of convex cones and zonotopes. It transpires that one can further simplify this and obtain a characterization which involves a single zonotope and a single cone.

Below, we write  $G(\pi)$  to denote the generator of the zonotope for the path  $\pi_0 \pi_1 \dots \pi_n$  as defined in Subsubsection 3.2.1; for each  $1 \leq i \leq n$ , we write  $G(\chi_i)$  to denote the generator of the convex cone for the cycle  $\chi_i$  as defined in Subsubsection 3.1.1.

► **Theorem 7.**  $\text{Reach}(\sigma) \stackrel{is^2}{=} \text{relint}(\text{zono}(G(\pi))) \cup \text{adj}(G(\pi)) + \text{relint}(\text{cone}(\bigcup_{i=1}^n G(\chi_i)))$ .

To prove the theorem, we establish two intermediate results. The first one, together with Theorem 5, already yields the first (Minkowski) summand. The result below follows immediately from the definitions and commutativity of the Minkowski sum.

► **Lemma 8.** *We have that  $\text{Reach}(\sigma) = \text{Reach}(\pi_0 \pi_1 \dots \pi_n) + \sum_{i=1}^n \text{Reach}(\chi_i^+)$ .*

The next result allows us to group the sums of cycle reachability sets into a single convex cone. Indeed, Theorem 1 and an induction on the following lemma yield the last summand from Theorem 7.

► **Lemma 9.** *Let  $C$  and  $C'$  be cones with generators  $G$  and  $G'$  respectively. Then,  $C + C' = \text{cone}(G \cup G')$  and  $\text{relint}(C + C') = \text{relint}(C) + \text{relint}(C')$ .*

<sup>2</sup> To avoid clutter, we omit some parentheses: union has higher precedence than Minkowski sum.

## 4 The Complexity of QVASS Reachability

In this section, we use our results concerning the geometry of reachability sets to give an NP decision procedure for the reachability problem.

► **Theorem 10.** *Given a QVASS of dimension  $d$ , and two configurations  $p(\mathbf{x})$  and  $q(\mathbf{y})$ , determining whether  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$  is in NP.*

First, without loss of generality, we assume  $\mathbf{x} = \mathbf{0}$ . Indeed,  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$  if and only if  $p(\mathbf{0}) \xrightarrow{*} q(\mathbf{y} - \mathbf{x})$ . In the following we prove that if  $p(\mathbf{0}) \xrightarrow{*} q(\mathbf{x})$  then there is an LPS  $\sigma$  such that  $p(\mathbf{0}) \xrightarrow{\pi} q(\mathbf{x})$  for some  $\pi \in \sigma$  and  $\sigma$  has size polynomial on the number of transitions  $|T|$  and on the dimension  $d$ . Then, we show that checking whether  $p(\mathbf{0}) \xrightarrow{*} q(\mathbf{x})$  under a given linear path scheme is decidable in polynomial time. It follows that to check whether a configuration is reachable, in a general QVASS, one can guess a polynomial-sized LPS and check whether the corresponding configuration is reachable in it.

### 4.1 Short linear path schemes suffice

Presently, we argue that for any path we can find an LPS with a number of cycles that is polynomial in the number of transitions of the QVASS and the dimension such that all paths and cycles are simple, the set of transitions in the LPS is the same as that in the path, and the reachability set of the LPS includes that of the path.

For convenience, we define the *support* of a path  $\pi = t_1 \dots t_n$  as the set of all transitions that are present in the path:  $\llbracket \pi \rrbracket = \{t_i \mid 1 \leq i \leq n\}$ . For an LPS  $\sigma = \pi_0 \chi_1^+ \dots \chi_n^+ \pi_n$ , its support is  $\llbracket \sigma \rrbracket = \llbracket \pi_0 \rrbracket \cup \bigcup_{i=1}^n (\llbracket \chi_i \rrbracket \cup \llbracket \pi_i \rrbracket)$ .

► **Theorem 11.** *Let  $\pi$  be an arbitrary path. Then, there exists a linear path scheme  $\sigma = \pi_0 \chi_1^+ \pi_1 \dots \chi_n^+ \pi_n$ , with  $n \leq |T|$ , such that all  $\pi_i$  and  $\chi_i$  are simple paths and cycles, respectively,  $\llbracket \pi \rrbracket = \llbracket \sigma \rrbracket$ , and  $\text{Reach}(\pi) \subseteq \text{Reach}(\sigma)$ .*

Most properties of the LPS in the result above follow from considering an LPS with minimal length, with the length of an LPS defined as  $|\sigma| = |\pi_0| + \sum_{i=1}^n |\chi_i| + |\pi_n|$ . The main technical hurdle is thus the upper bound on the number of cycles. Our approach is to remove cycles whose support is covered by other cycles. The result below, which follows directly from Theorem 1 and Lemma 9, gives us that flexibility. As in Subsection 3.3, we write  $G(\chi)$  to denote the generator of the convex cone for  $\chi$ , i.e.  $G(\chi) = \{\ell(t) \mid t \in \llbracket \chi \rrbracket\}$ .

► **Lemma 12.** *Let  $\chi$  be a cycle and  $C$  be a set of cycles. If  $\llbracket \chi \rrbracket \subseteq \bigcup_{\theta \in C} \llbracket \theta \rrbracket$  then  $\text{Reach}(\chi^+) + \sum_{\theta \in C} \text{Reach}(\theta^+) = \sum_{\theta \in C} \text{Reach}(\theta^+)$ .*

Hence, to check whether a configuration is reachable in a general QVASS, one can guess a polynomial-sized LPS and check whether the corresponding configuration is reachable in it. To conclude membership in NP, it remains to argue that the latter check can be realized in polynomial time.

### 4.2 Reachability for linear path schemes is tractable

In this section, we show that determining whether a configuration is reachable via a linear path scheme is decidable in polynomial time.

► **Theorem 13.** *Given LPS  $\sigma$  and  $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^d$ , determining whether  $\mathbf{y} \in \text{Reach}^{\mathbf{x}}(\sigma)$  is in P.*

Based on Theorem 7 and the geometric characterizations of the reachability sets of cycles and paths, it suffices to show how to determine whether there exist  $\mathbf{z}, \mathbf{c} \in \mathbb{Q}^d$  in a zonotope and a cone, respectively, such that  $\mathbf{y} = \mathbf{z} + \mathbf{c}$ . To do so, we make Lemma 6 and Lemma 2 effective by encoding them into systems of linear inequalities with *strict-inequality constraints*. It is known that the feasibility problem for linear programs is decidable in polynomial time even in the presence of strict inequalities (see, e.g., [18, Ch. 8.7.1]).

Henceforth, we fix an LPS  $\sigma = \pi_0 \chi_1^+ \pi_1 \dots \chi_n^+ \pi_n$ . We also adopt the notation from Subsection 3.3:  $G(\pi)$  denotes the generator of the zonotope for the path  $\pi_0 \pi_1 \dots \pi_n$ ; and  $G(\chi_i)$  the generator of the cone for  $\chi_i$  for each  $1 \leq i \leq n$ .

#### 4.2.1 Encoding the zonotope

Let  $G(\pi) = \{\mathbf{g}_1, \dots, \mathbf{g}_m\}$ . We now define the matrix  $\mathbf{A} \in \mathbb{Q}^{d \times (m+d)}$  and the vector  $\mathbf{a} \in \mathbb{Q}^d$ :

$$\mathbf{A} = \begin{pmatrix} (\mathbf{g}_1)_1 & (\mathbf{g}_2)_1 & \dots & (\mathbf{g}_m)_1 & -1 & 0 & \dots & 0 \\ (\mathbf{g}_1)_2 & (\mathbf{g}_2)_2 & \dots & (\mathbf{g}_m)_2 & 0 & -1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (\mathbf{g}_1)_d & (\mathbf{g}_2)_d & \dots & (\mathbf{g}_m)_d & 0 & \dots & 0 & -1 \end{pmatrix} \text{ and } \mathbf{a} = (0, \dots, 0), \quad (1)$$

Further, we define the matrix  $\mathbf{B} \in \mathbb{Q}^{m \times (m+d)}$  and the vector  $\mathbf{b} \in \mathbb{Q}^m$  as:

$$\mathbf{B} = (\mathbf{I} \ \mathbf{0} \ \dots \ \mathbf{0}) \text{ and } \mathbf{b} = (1, 1, \dots, 1), \quad (2)$$

where  $\mathbf{I}$  is the  $m \times m$  identity matrix and  $\mathbf{0}$  is the  $m \times 1$  zero vector. Finally, we define  $\mathbf{C} \in \mathbb{Q}^{m \times (m+d)}$  and  $\mathbf{c} \in \mathbb{Q}^m$  as follows.

$$\mathbf{C} = (-\mathbf{I} \ \mathbf{0} \ \dots \ \mathbf{0}) \text{ and } \mathbf{c} = (0, 0, \dots, 0) \quad (3)$$

► **Lemma 14.**  $A\mathbf{z} = \mathbf{a} \wedge B\mathbf{z} \leq \mathbf{b} \wedge C\mathbf{z} < \mathbf{c}$  has a solution  $(\boldsymbol{\alpha}, \mathbf{y}) \in \mathbb{Q}^{m+d}$  iff  $\mathbf{y} \in \text{Reach}(\pi)$ .

This follows immediately from the fact that, by construction, the system has a solution if and only if there exists  $(\alpha_1, \dots, \alpha_m) \in (0, 1]$  such that  $\mathbf{y} = \sum_{i=1}^m \alpha_i \mathbf{g}_i$ , and Lemma 6.

#### 4.2.2 Encoding the cone

The encoding for the cone is even simpler, but requires we adapt our notation slightly.

Let  $\bigcup_{i=1}^n G(\chi_i) = \{\mathbf{g}_1, \dots, \mathbf{g}_m\}$ . We define the matrix  $\mathbf{A} \in \mathbb{Q}^{d \times (m+d)}$  and vector  $\mathbf{a} \in \mathbb{Q}^d$  as in Equation 1; and  $\mathbf{C} \in \mathbb{Q}^{m \times (m+d)}$  and  $\mathbf{c} \in \mathbb{Q}^d$  as in Equation 3.

► **Lemma 15.**  $A\mathbf{z} = \mathbf{a} \wedge C\mathbf{z} < \mathbf{c}$  has a solution  $(\boldsymbol{\alpha}, \mathbf{y}) \in \mathbb{Q}^{m+d}$  iff  $\mathbf{y} \in \sum_{i=1}^n \text{Reach}(\chi_i^+)$ .

This time the lemma follows from Lemma 2 and because, by construction, the system has a solution if and only if there are  $(\alpha_1, \dots, \alpha_m)$  nonnegative such that  $\mathbf{y} = \sum_{i=1}^m \alpha_i \mathbf{g}_i$ .

**Proof of Theorem 13.** The result follows from the fact that Lemma 8 can be made effective by encoding it into a master system of linear inequalities for the zonotope and the cycle. ◀

## 5 The Complexity of $\mathbb{Q}_{\geq 0}$ VASS Reachability

We now give an NP decision procedure for the reachability problem for  $\mathbb{Q}_{\geq 0}$  VASS.

► **Theorem 16.** Given a  $\mathbb{Q}_{\geq 0}$  VASS of dimension  $d$ , and configurations  $p(\mathbf{x})$  and  $q(\mathbf{y})$ , determining whether  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$  is in NP.

## 11:10 The Geometry of Continuous VASS

The structure of our argument is similar to the one in Section 4: we prove that short LPS suffice and then we prove reachability is tractable for LPS. The first part is considerably more complicated for  $\mathbb{Q}_{\geq 0}$ VASS and it is based on the fact that short LPS exist for  $\mathbb{Q}$ VASS. For this reason, we need additional notation: We write  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$  to denote that  $q(\mathbf{y})$  is reachable from  $p(\mathbf{x})$  in a  $\mathbb{Q}_{\geq 0}$ VASS and instead use  $p(\mathbf{x}) \xrightarrow{-*} q(\mathbf{y})$  to denote that  $q(\mathbf{y})$  is reachable from  $p(\mathbf{x})$  with respect to  $\mathbb{Q}$ VASS semantics (i.e. when allowing negative counter values). Similarly, we write  $\text{Reach}^{\mathbf{x}}(\cdot)$  for reachability sets w.r.t.  $\mathbb{Q}$ VASS and  $\text{Reach}_{\geq 0}^{\mathbf{x}}(\cdot)$  for that w.r.t.  $\mathbb{Q}_{\geq 0}$ VASS. We make repeated use of the following result by Blondin and Haase.

► **Lemma 17** (From [2, Proposition 4.5]). *There exists a path  $\pi$  such that  $q(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$  if and only if there exist paths  $\pi_1, \pi_2, \pi_3$  such that:*

1.  $q(\mathbf{x}) \xrightarrow{\pi_2} q(\mathbf{y})$ ;
2.  $q(\mathbf{x}) \xrightarrow{\pi_1} q(\mathbf{x}')$ , for some  $\mathbf{x}' \in \mathbb{Q}_{\geq 0}^d$ ;
3.  $q(\mathbf{y}') \xrightarrow{\pi_3} q(\mathbf{y})$ , for some  $\mathbf{y}' \in \mathbb{Q}_{\geq 0}^d$ ; and
4.  $\llbracket \pi \rrbracket = \llbracket \pi_1 \rrbracket = \llbracket \pi_2 \rrbracket = \llbracket \pi_3 \rrbracket$ .

Moreover, if item 1–item 4 hold, then  $\mathbf{y} \in \text{Reach}_{\geq 0}^{\mathbf{x}}(\pi_1 \pi_2^+ \pi_3)$ .

Intuitively,  $q(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$  if the following conditions hold: first, we obviously need  $q(\mathbf{x}) \xrightarrow{\pi_2} q(\mathbf{y})$ , and second, we need  $q(\mathbf{x})$  to allow some “wiggle room” using the same transitions as  $\pi$  and while keeping the counters nonnegative. Similarly, there should be wiggle room to reach  $q(\mathbf{y})$  with nonnegative counters. The lemma also shows that these conditions are necessary.

### 5.1 Short linear path schemes suffice

We start by proving that short LPS suffice.

► **Theorem 18.** *Let  $\pi$  be an arbitrary path such that  $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ . Then, there exists a linear path scheme  $\sigma = \pi_0 \chi_1^+ \pi_1 \dots \chi_n^+ \pi_n$ , with:*

- $n \leq |Q|$ ,
- $\pi_i$  is a simple path for all  $0 \leq i \leq n$ ,
- $|\chi_i| \leq 4|Q|(|T| + d + 2)(|T| + 1)$  for all  $1 \leq i \leq n$ , and
- $\mathbf{y} \in \text{Reach}_{\geq 0}^{\mathbf{x}}(\sigma)$ .

Let us we outline our approach to prove this. Consider a path  $\pi$  such that  $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ . We decompose  $\pi$  into  $\pi = \tau_0 \theta_1 \tau_1 \dots \theta_n \tau_n$  where the  $\tau_i$  are simple paths separated by at most  $|Q|$  (not necessarily simple) cycles  $\theta_i$ . We would now like to replace each cycle  $\theta$  with a short LPS  $\chi^+$ , as per the third item in Theorem 18. Note that we cannot readily do so using Theorem 11, as it does not guarantee nonnegative reachability. We thus take a more elaborate approach, as follows.

Consider a cycle  $\theta$ . By applying Lemma 17, we can replace  $\theta$  by an LPS  $\pi_1 \pi_2^+ \pi_3$ . We now apply Theorem 11 to  $\pi_2$ , thus obtaining an LPS  $\sigma = \mu_0 \zeta_1^+ \mu_1 \dots \zeta_m^+ \mu_m$  such that  $\text{Reach}^z(\pi_2) \subseteq \text{Reach}^z(\sigma)$  for all  $z$  (note that nonnegativity is no longer maintained). Recall that our goal is to represent  $\pi_2^+$  as an LPS, so we cannot use  $\sigma^+$ , as it is not an LPS. Instead, we show that by looking at the concrete path  $\bar{\sigma} = \mu_0 \zeta_1 \mu_1 \dots \zeta_m \mu_m$  we have  $\text{Reach}^z(\pi_2^+) \subseteq \text{Reach}^z(\bar{\sigma}^+)$ , and  $\bar{\sigma}^+$  is an LPS.

The next challenge is to plug back  $\bar{\sigma}^+$  as part of an LPS for  $\theta$ . To do so, we need to find LPS for  $\pi_1$  and  $\pi_3$ . We show that this is possible. We can now use Lemma 17 again, in the opposite direction, to conclude that  $\pi_1 \pi_2^+ \pi_3$  can be described by an LPS with appropriate bounds, retaining nonnegative reachability.

## 5.2 Reachability for linear path schemes is tractable

As in the QVASS case, here we are able to prove that determining whether a configuration is reachable via an LPS is decidable in polynomial time.

► **Theorem 19.** *Given LPS  $\sigma$  and  $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_{\geq 0}^d$ , determining if  $\mathbf{y} \in \text{Reach}_{\geq 0}^{\mathbf{x}}(\sigma)$  is in  $\mathbf{P}$ .*

Once more, our argument relies on an encoding into a system of linear inequalities. However, in contrast to QVASS, the encoding is slightly less elegant. For each path  $\pi = (p_1, p_2) \dots (p_m, p_{m+1})$  occurring inside  $\sigma$ , instead of encoding an (affine) zonotope into a system of linear equalities, we focus directly on the  $m$  steps of the path.

Let  $\mathbf{b}_i = \ell(p_i, p_{i+1})$  for all  $1 \leq i \leq m$ . We will use the  $\mathbf{b}_i$  instead of the zonotope basis  $G(\pi)$  of Subsection 4.2. We now adapt the system from Subsection 4.2 to account for the nonnegativity of partial sums induced by the path prefixes. Recall  $\mathbf{A}, \mathbf{a}, \mathbf{B}, \mathbf{b}, \mathbf{C}$ , and  $\mathbf{c}$  as defined in Equation 1, Equation 2, and Equation 3. We introduce  $d + md$  new variables – the first  $d$  account for an initial vector  $\mathbf{x}$ , and the remaining  $md$  represent the intermediate values of the path after each transition.

Intuitively, we obtain from the above matrices new ones, denoted  $\mathbf{A}', \mathbf{a}', \mathbf{B}', \mathbf{b}', \mathbf{C}'$  so that  $\mathbf{A}'$  includes the constraints  $\bigwedge_{k=1}^d \bigwedge_{n=1}^m \left( \mathbf{x}_k + \sum_{j=1}^n (\mathbf{b}_j)_k z_j \right) = z_{m+2d+(n-1)d+k} \geq 0$ . We thus have the following.

► **Lemma 20.** *For a path  $\pi$ , the system  $\mathbf{A}'\mathbf{z} = \mathbf{a}' \wedge \mathbf{B}'\mathbf{z} \leq \mathbf{b}' \wedge \mathbf{C}'\mathbf{z} < \mathbf{c}'$  of linear inequalities has a solution  $(\boldsymbol{\alpha}, \mathbf{y}, \mathbf{x}, \boldsymbol{\iota}) \in \mathbb{Q}^{m+2d+md}$  if and only if  $p_1(\mathbf{x}) \xrightarrow{\pi} p_{m+1}(\mathbf{y})$ .*

**Proof of Theorem 19.** By Lemma 20, it suffices to argue that nonnegative reachability via a cycle  $\chi^+$  of  $\sigma$  can also be encoded into a system of linear inequalities. For this, we make use of the “if” direction of Lemma 17, which for  $\chi^+$  amounts to  $p(\mathbf{x}) \xrightarrow{\chi^+} p(\mathbf{y})$  iff (1)  $p(\mathbf{x}) \xrightarrow{\chi^+} p(\mathbf{y})$ , (2)  $p(\mathbf{x}) \xrightarrow{\chi} p(\mathbf{x}')$  for some  $\mathbf{x}' \in \mathbb{Q}_{\geq 0}$ , (3)  $p(\mathbf{y}') \xrightarrow{\chi} p(\mathbf{y})$  for some  $\mathbf{y}' \in \mathbb{Q}_{\geq 0}$ . Condition 1 can be encoded in a system of linear inequalities by Lemma 15, and conditions 2 and 3 can be encoded in such systems too as per Lemma 20. We can now conjoin the systems for the paths and cycles to obtain a master system of linear inequalities of polynomial size. ◀

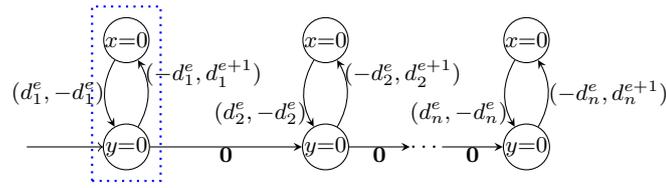
## 6 The Complexity of Reachability with Zero Tests

We now argue that the reachability problem for continuous VASS with *zero tests* is **NP**-hard, already for LPS of dimension 2. For convenience, we state the result for QVASS. However, we note that the same proof establishes the result for  $\mathbb{Q}_{\geq 0}$ VASS.

We start by formally defining the model. A continuous VASS  $\mathcal{V}$  of dimension 2 with zero tests is a tuple  $(Q, t, \ell, Z_1, Z_2)$ , where  $\mathcal{V}' = (Q, t, \ell)$  is a continuous VASS and  $Z_i \subseteq Q$  for  $i = 1, 2$ . A run  $\rho = q_1(\mathbf{x}_1) \dots q_n(\mathbf{x}_n)$  of such a VASS is a run of  $\mathcal{V}'$  such that, additionally, for all  $1 \leq i \leq n$  we have that if  $q_i \in Z_j$ , for some  $j \in \{1, 2\}$ , then  $(\mathbf{x}_i)_j = 0$ . That is, any run that reaches a state in  $Z_j$  must be such that the value of the  $j$ -th counter is 0 then.

► **Theorem 21.** *For every  $d \in \mathbb{N}, d \geq 2$ , given a QVASS (or  $\mathbb{Q}_{\geq 0}$ VASS) of dimension  $d$ , and two configurations  $p(\mathbf{x})$  and  $q(\mathbf{y})$ , determining whether  $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$  is **NP**-hard, even for linear path schemes of dimension 2.*

We reduce from the PRIMECOVER problem: Given a set  $X$  of prime numbers and a collection  $S$  of subsets of  $X$ , with  $T \in \mathbb{N}$ , determine whether there is a subset  $S' \subseteq S$  such that  $\prod_{s' \in S'} \prod_{p \in s'} p = T$ . It is straightforward to prove PRIMECOVER is **NP**-hard by reduction from the the EXACTCOVER problem [13].



■ **Figure 3** In the dotted blue box, a multiplier gadget is shown: the state above is an element of  $Z_1$  (noted by  $x = 0$ ) while the state below is an element of  $Z_2$  (noted by  $y = 0$ ); The whole LPS encodes an instance of PRIMECOVER with  $S = \{s_1, s_2, \dots, s_n\}$  – recall that  $d_i = \prod_{p \in c_i} p$ .

Now, for each  $s \in S$  we create multiplier gadgets as depicted in Figure 3 where  $d = \prod_{p \in s} p$  and  $e = \lceil \log_2(\prod_{s \in S} \prod_{p \in s} p) \rceil + 1$ , and we link them in an LPS with transitions  $(q_i, q_{i+1})$ , for  $1 \leq i \leq |S|$ , labelled with  $(0, 0)$  updates (see Figure 3). We claim that PRIMECOVER has a positive answer if and only if  $q_1(1, 0) \xrightarrow{*} q_{|S|}(T, 0)$  in the constructed LPS.

## 7 Conclusion

We gave geometrical characterizations for the reachability sets of continuous VASS and their flat and LPS restrictions. Using these, we showed that polynomial-sized LPS suffice as witnesses of reachability and that reachability in linear path schemes is tractable. In addition, we sharpened hardness results in the presence of zero tests: it is **NP**-hard already for dimension two.

## References

- 1 Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazic, Pierre McKenzie, and Patrick Totzke. The reachability problem for two-dimensional vector addition systems with states. *J. ACM*, 68(5):34:1–34:43, 2021. doi:10.1145/3464794.
- 2 Michael Blondin and Christoph Haase. Logics for continuous reachability in Petri nets and vector addition systems with states. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.
- 3 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. Reachability in fixed dimension vector addition systems with states. 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 48:1–48:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CONCUR.2020.48.
- 4 Wojciech Czerwinski, Slawomir Lasota, Christof Löding, and Radoslaw Piórkowski. New pumping technique for 2-dimensional VASS. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 62:1–62:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.62.
- 5 Wojciech Czerwinski, Slawomir Lasota, and Lukasz Orlikowski. Improved lower bounds for reachability in vector addition systems. 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 128:1–128:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.128.

- 6 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.
- 7 René David and Hassane Alla. *Discrete, continuous, and hybrid Petri nets*, volume 1. Springer, 2010.
- 8 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPICs*, pages 31:1–31:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.FSTTCS.2018.31.
- 9 Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, UK, 2012. URL: <http://ora.ox.ac.uk/objects/uuid:f43bf043-de93-4b5c-826f-88f1bd4c191d>.
- 10 Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. doi:10.1145/3242953.3242964.
- 11 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems – 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2\_9.
- 12 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 – Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8\_25.
- 13 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 14 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.
- 15 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.
- 16 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 – Concurrency Theory, 15th International Conference, London, UK, August 31 – September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004. doi:10.1007/978-3-540-28644-8\_26.
- 17 Marvin Lee Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.
- 18 Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
- 19 Sylvain Schmitz. The complexity of reachability in vector addition systems. *ACM SigLog News*, 3(1):4–21, 2016.



# Competitive Search in the Line and the Star with Predictions

Spyros Angelopoulos  

CNRS and LIP6, Sorbonne University, Paris, France

---

## Abstract

We study the classic problem of searching for a hidden target in the line and the  $m$ -ray star, in a setting in which the searcher has some *prediction* on the hider's position. We first focus on the main metric for comparing search strategies under predictions; namely, we give positive and negative results on the *consistency-robustness* tradeoff, where the performance of the strategy is evaluated at extreme situations in which the prediction is either error-free, or adversarially generated, respectively. For the line, we show tight bounds concerning this tradeoff, under the *untrusted advice* model, in which the prediction is in the form of a  $k$ -bit string which encodes the responses to  $k$  binary queries. For the star, we give tight, and near-tight tradeoffs in the *positional* and the *directional* models, in which the prediction is related to the position of the target within the star, and to the ray on which the target hides, respectively. Last, for all three prediction models, we show how to generalize our study to a setting in which the performance of the strategy is evaluated as a function of the searcher's desired tolerance to prediction errors, both in terms of positive and inapproximability results.

**2012 ACM Subject Classification** Theory of computation → Online algorithms; Theory of computation → Theory and algorithms for application domains

**Keywords and phrases** Search problems, line and star search, competitive ratio, predictions, consistency and robustness

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.12

**Funding** This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

**Acknowledgements** I am thankful to Shahin Kamali for several helpful discussions.

## 1 Introduction

Searching for a hidden target is one of the original disciplines within the field of Operations Research, but also a topic of significant study in Computer Science, both from the point of view of theoretical analysis and applications. This class of problems typically involves a mobile *searcher* that must locate an immobile *target* (often called *hider*) which hides in some unknown point of the search *environment*. Search problems provide natural formulations of real-life applications such as search-and-rescue missions [41], de-mining operations [2], and robot-based exploration [27].

Among the most well-studied search problems is *searching on the line*, in which the environment is the unbounded line, and its generalization, the  *$m$ -ray search*, or *star search* problem. In the  *$m$ -ray search* problem, the environment consists of  $m$  unbounded and concurrent rays, with a common point  $O$ , which is called the *origin*. Starting from  $O$ , the searcher must locate the target by following a *strategy*, defined as an infinite sequence of the form  $(x_i, u_i)_i$ , where  $x_i \in \mathbb{R}^+$  and  $u_i \in \{0, \dots, m-1\}$ , and with the following semantics: in iteration  $i$ , the searcher starts from  $O$ , traverses the ray  $u_i$  up to distance  $x_i$  from  $O$ , then returns back to  $O$ , before continuing with iteration  $i+1$ , until the target is eventually located. Note that for  $m=2$ , the star environment reduces to the infinite line.



© Spyros Angelopoulos;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 12; pp. 12:1–12:15

Leibniz International Proceedings in Informatics



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

Since the search environment is unbounded, the standard framework for evaluating the performance of a search strategy is *competitive analysis*, first introduced in [11]. Given a target  $t$  hiding at some unknown point of the star, define  $d(t)$  as the distance of  $t$  from  $O$ , and  $d(X, t)$  as the distance covered (or *cost* incurred) by a searcher that follows  $X$ , until  $t$  is located (i.e., the first time the searcher is reached, assuming a unit-speed searcher). The *competitive ratio* of  $X$  is formally defined as

$$\text{Cr}(X) = \sup_t \frac{d(X, t)}{d(t)}. \quad (1)$$

Searching on the line has a long history of study, going back to the work of Bellman [12] and Beck [10]. Beck and Newman [11] were the first to show that an optimal competitive ratio equal to 9 can be obtained by a simple doubling strategy, i.e., a strategy of the form  $x_i = 2^i$ . The  $m$ -ray search problem was first studied in the seminal works of Gal [24, 25] and independently by Baeza-Yates *et al.* [8]. Both problems have been extended in a variety of settings and generalizations related to TCS, AI and OR since the 1960s, due to their useful abstraction of resource allocation under uncertainty. For instance, linear and ray searching have connections to the design of *interruptible systems* in AI [4, 13], the design of *hybrid* algorithms [32], and pipelined *filter ordering* in databases [18]. They are also involved in the analysis of strategies for more complex search problems, such as *spiral* search on the plane [38]. There are numerous studies on variants of linear and star search; see, e.g., [31, 36, 29, 44, 33, 16, 20, 5, 19, 37, 15, 34, 46, 6, 43, 51, 36, 17] as well as the book [1] for a game-theoretic perspective of these problems. Note that the above are only some representative works, and that the problems have been studied under several other variants.

## 1.1 Searching with predictions

In this work, we study the power and limitations of search strategies with *predictions*, in which the searcher aims to improve the competitive ratio of its strategy by leveraging some inherently imperfect information on the target. This follows a very active line of research in online computation and algorithms with incomplete information, that was initiated with the works [45] and [49]. A very large number of problems have been studied under this model (see, e.g., the survey [47] and the online collection [42]).

In regards to the search problems we study, the nature of the prediction may vary according to the application at hand. We are interested in the following models, which were introduced in [3] in the context of linear search.

- (a) The prediction is a *k-bit string*. Here, the search is enhanced with a  $k$ -bit string that encodes some information on the target; alternatively, we may think of the prediction string as responses to binary *queries* given by  $k$  experts. For example, a single bit can provide a (potentially erroneous) response to queries such as “Is the target at distance at most  $d$  from  $O$ ”, or “Is the target on an even-indexed ray?”. This is a powerful model that generalizes the concept of *advice complexity* so as to allow for advice that may be erroneous. Note that search and exploration problems have been studied extensively under the standard advice complexity model (see, e.g., [21, 23, 28, 35, 48]), however all such studies rely crucially on advice that is error-free. Moreover, unlike works in which each query is *noisy* [14], i.e., the query responses are erroneous with some known probability, in our setting we do not rely on any probabilistic assumptions in regards to the quality of the advice.

- (b) The prediction is *directional*. Here, the prediction is an index in  $\{0, \dots, m - 1\}$  which describes the ray on which the target lies. This is a natural prediction can be useful, for example, in a search-and-rescue application, in which there is a hint about the direction a missing person may have taken when last seen.
- (c) The prediction is *positional*. Here, the prediction describes the position of the target within the environment, namely it is of the form  $(d, u)$ , where  $d$  corresponds to the predicted distance from the origin, and  $u$  corresponds to the predicted ray on which the target hides. This is, likewise, a very natural prediction (e.g., in a search-and-rescue mission, it provides a hint about the last reported whereabouts of the missing person).

We establish two objectives towards the evaluation of search strategies with predictions. The first objective is to find strategies of optimal, or near-optimal tradeoff between their *consistency* (namely, the competitive ratio assuming error-free prediction) and their *robustness* (namely, the competitive ratio assuming adversarially generated predictions). This is one of the standard methods of analyzing algorithms with predictions, since it establishes strong guarantees on worst-case (extreme) situations with respect to the quality of the prediction; see, e.g., [52, 40, 39, 3, 7] for applications to other online problems, and settings of incomplete information, more generally. Specifically, we are interested in showing both positive and negative results on the best-possible consistency that can be achieved by  $r$ -robust strategies, for any given  $r$ .

Our second objective goes beyond the consistency/robustness tradeoffs, and we evaluate the performance of the search strategy beyond the two extreme scenarios of error-free and adversarial error. Specifically, we study the novel setting in which the searcher defines an application-specific *tolerance* parameter  $H$  that determines its desired tolerance to errors or, equivalently, an anticipated upper bound on the prediction error (that may be known by historical data on previous searches). This parameter is defined appropriately for each of the three prediction models we study. Namely, in the untrusted advice model,  $H$  is related to the number of erroneous advice bits (or query responses); in the directional model  $H$  describes the distance of the predicted ray index to the one of the actual hiding target; and in the positional model,  $H$  is related to the distance between the predicted and the actual target position. The tolerance model is motivated by recent works in learning-enhanced online algorithms with *weak predictions*, in which the prediction is an upper bound of some pertinent parameter of the input (see e.g., online knapsack with frequency predictions [30], where the prediction is an upper bound on the size of items that appear online). Our objective is thus to quantify the tradeoff between the competitive ratio and the robustness as a function of the tolerance and other parameters of the problem (e.g., the number of queries, in the query-based model). Following [30], we will make use of the term *weak prediction* to refer to this setting.

The problems we study have applications in more general decision-making settings that go beyond the confines of search theory. This is since  $m$ -ray search, as discussed above, is an abstraction of resource allocation among  $m$  different tasks. To illustrate with an example, consider a researcher who has to allocate time among  $m$  different projects, without knowing ahead of time which project will be completed successfully. The researcher, however, may have some intuition about which of these tasks is the most likely to succeed. This problem fits the  $m$ -ray search abstraction with a directional prediction. In the weak prediction setting,  $H$  describes, more generally, the specific projects which the researcher believes are more likely to be successful.

Learning-augmented search has received attention in recent years. [3] studied consistency/robustness (Pareto) tradeoffs for linear search in the three prediction models described above. [9] studied a graph search setting where every node in the graph provides a prediction of its distance to the target vertex. [22] showed how to robustify graph exploration algorithms, where the prediction is related to the spanning tree of the explored graph.

## 1.2 Contribution

Our first results apply to the untrusted advice model (i.e, the  $k$ -query model). We prove tight upper and lower bounds on the best consistency that an  $r$ -robust strategy for linear search can achieve, for any  $r \geq 9$ , any size of advice  $k$ , and with no assumptions on the nature of the strategy. This improves upon both the upper and the lower bounds of [3], which gave a non-tight lower bound for  $k = 1$  and  $r = 9$ , and a non-tight lower bound for  $r > 9$  and  $k = 1$  for a restricted class of strategies called *asymptotic*. Here, the challenge is on the lower bound side. Specifically, we reduce the problem to a *parallel search* problem that involves  $2^k$  searchers, and we rely on a novel application of Gal’s functional theorem [26] to prove an information-theoretic tight lower bound. While this theorem has been previously applied in parallel search problems [44], its application in our setting is much more challenging, since we require that each of the  $2^k$  searchers must be individually  $r$ -robust. Specifically, unlike previous works, the proof requires an explicit *labeling* scheme that maps the search lengths of each parallel searcher to lengths of a “global” sequence. We also extend our upper bound to weak predictions, by applying tools from the theory of games with a *lying responder* [50], in order to bound the effect of erroneous query responses to the performance.

Our second class of results is on the directional prediction model of  $m$ -ray search. We give the first upper and lower bounds on the consistency-robustness tradeoffs, which extend those of [3] to star search. Here, the main challenge is again on the lower bound side, and specifically in the weak predictions setting. We show how a generalization of a biased search approach, in which the searcher allocates more time towards the predicted ray, allows us to prove an asymptotically tight bound on the competitive ratio as a function of the tolerance and the number of rays.

Last, we show tight (Pareto-optimal) consistency-robustness tradeoffs for  $m$ -ray search in the positional model. As with the directional model, the only previous known results applied to linear search [3]. The proof uses tools that circumvent the exact study of linear recurrence relations inherent in  $m$ -ray search problems. To our knowledge, this is a new approach towards impossibility results on this type of search games. As with the other models, we also provide tight upper and lower bounds on the competitive ratio under weak predictions. We emphasize that beyond the tight and near-tight results, the generalization to star search and the accompanied analysis under weak predictions are conceptually novel aspects of this work and extend the performance guarantees beyond the consistency/robustness tradeoffs.

Due to space limitations, we omit or sketch technical details in some of the proofs.

## 2 Preliminaries

We review some notation and known results concerning  $m$ -ray searching. Without predictions, a strategy is described by a sequence of the form  $X = (x_i, u_i)_{i \geq 0}$ ; we refer to  $i$  as the *iteration*<sup>1</sup> of the strategy, to  $x_i$  as the *length* of the  $i$ -th search *segment*, to  $u_i$  as the ray searched in iteration  $i$ , and to the point at which the searcher turns in the  $i$ -th iteration as the

---

<sup>1</sup> We will consider a numbering of iterations that starts either with 0, or with 1, depending on which simplifies the presentation.

corresponding *turn point*. Note that for linear search ( $m = 2$ ), we may assume, without loss of generality, that  $u_i = i \bmod m$ , and that  $x_{i+2} > x_i$ . We make the standing assumption that the target lies within distance at least a fixed value, otherwise every strategy has unbounded competitive ratio. It is well-known that the worst-case hiding positions of the target, i.e., the positions that maximize  $\text{Cr}(X)$ , are infinitesimally beyond the turn points of a searcher that follows  $X$ , namely, at distances  $x_i + \epsilon$  on rays  $u_i$ , for  $\epsilon > 0$ .

A strategy for  $m$ -ray search is called *cyclic*, if it explores the rays in a fixed permutation of  $\{0, \dots, m-1\}$ , e.g., if  $u_i = i \bmod m$ . The competitive ratio of a cyclic strategy of the form  $X = (x_i, i \bmod m)_i$  is easily shown to be equal to

$$\text{Cr}(X) = 1 + \sup_i \frac{2 \sum_{j=0}^{i+m-1} x_j}{x_i}. \quad (2)$$

In particular, a cyclic strategy is called *geometric* if  $x_i = b^i$ , for some fixed  $b > 1$  which is called the *base* of the strategy; we will denote such strategies by  $G_b$ . Geometric strategies are significant since they are often optimal for several variants of linear search. From (2), it follows that the competitive ratio of  $G_b$  is therefore equal to  $1 + 2b^m/(b-1)$ . This expression is minimized for  $b = m/(m-1)$ , and the resulting optimal competitive ratio, denoted by  $r_m^*$ , is equal to

$$1 + 2\rho_m^*, \quad \text{where } \rho_m^* = \frac{m^m}{(m-1)^{m-1}}.$$

Thus, given  $r \geq r_m^*$ , strategy  $G_b$  has competitive ratio at most  $r$  if  $b^m/(b-1) \leq \rho_r$ , where  $\rho_r$  is defined to be equal to  $(r-1)/2$ . We will denote by  $b_r$  the largest such  $b$  for which  $G_b$  is  $r$ -competitive, i.e., the largest real root of the function  $f(x) = x^m/(m-1) - \rho_r$ .

Under the prediction framework, the searcher is given some information  $h$  in regards to the target  $t$ , and determines a strategy  $X_h$  (we will often omit  $h$  when it is clear from context). Following [3], we define the *consistency* of a strategy as its competitive ratio assuming no prediction error, and its *robustness* as its competitive ratio assuming adversarial prediction error. Note, in particular, that the robustness of a strategy is equal to its competitive ratio without any prediction, and we will thus use these two terms interchangeably. We say that a strategy is  *$r$ -robust* if its robustness is at most  $r$  (similarly for the consistency), and that it is *Pareto-optimal* if its consistency and robustness are in an optimal tradeoff relation.

Let  $Y = (y_i)_{i=0}^\infty$  denote a sequence in  $\mathbb{R}^+$ . We define  $\alpha_Y$  as  $\alpha_Y = \limsup_{n \rightarrow \infty} y_n^{1/n}$ . This parameter appears prominently in Gal's theorem [26] which, informally, gives a lower bound on the supremum of a set of functionals by the supremum of these functionals over geometrically increasing sequences. From it, it follows that any  $m$ -ray search strategy  $Y$  with search lengths  $(y_i)_i$  has competitive ratio at least  $1 + 2 \frac{\alpha_Y^m}{\alpha_Y - 1}$ , hence if the strategy is  $r$ -competitive it must be that  $\alpha_Y \leq b_r$ .

### 3 Linear search with untrusted advice

In this section, we study linear search in a model in which the prediction is an untrusted advice string of size  $k$ . We first show optimal upper and lower bounds on the best consistency of  $r$ -robust strategies, then in Section 3.1 we study the extension to weak predictions.

Our results will show and exploit connections between a single-searcher strategy with  $k$ -bit advice, and a *multi-searcher* strategy with  $2^k$  parallel searchers, but no advice. Hence, we first present some definitions and notation concerning the setting of  $p > 1$  parallel searchers, labeled from the set  $\{0, \dots, p-1\}$ . In a  $p$ -searcher strategy, each searcher  $j$  defines its

own strategy of the form  $X_j = (x_{j,i}, u_{j,i})_{i=0}^{\infty}$ . We thus denote the  $p$ -searcher strategy as  $\mathcal{X} = \{X_j\}_{j=0}^{p-1}$ , or equivalently, we say that it is defined by the set  $\{X_j\}_{j=0}^{p-1}$ . The competitive ratio of a  $p$ -searcher strategy is the worst-case ratio of the first time one of the  $p$  searchers finds the target  $t$  (assuming unit-speed searchers) and the distance  $d(t)$  of the target from the origin [44].

Observe that the optimal consistency of an  $r$ -robust strategy with  $k$  advice bits is equal to the competitive ratio of a parallel search strategy that is defined by  $2^k$  searchers, each of which is individually  $r$ -robust. Namely, if the advice is error-free, it can be used to select the single-searcher strategy, among the  $2^k$  ones, that reaches the target at optimal cost. Note that, by construction, the robustness of this strategy is at most  $r$ , since each individual searcher is  $r$ -robust. This observation applies to both positive and negative results on the consistency/robustness tradeoffs.

We first show an upper bound on the consistency of  $r$ -robust strategies:

► **Theorem 1.** *For any  $r \geq 9$ , there is an  $r$ -robust strategy for searching on the line with  $k$ -bit advice that has consistency at most  $1 + 2 \frac{b_r^{1/q}}{b_r - 1}$ , where  $q = 2^{k-1}$ .*

**Proof sketch.** Let  $\mathcal{S}$  denote the  $2^k$ -parallel strategy as defined by the set  $S_0, \dots, S_{2^k-1}$  where

$$S_j = (b^{j+iq}, i \bmod 2), \text{ if } j \text{ is even and } S_j = (b^{j+iq}, (i+1) \bmod 2), \text{ if } j \text{ is odd,}$$

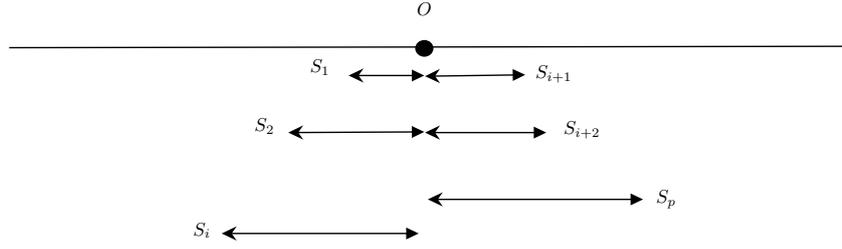
for some  $b > 1$  that will be specified later. That is, each individual strategy is near-geometric, and half of the searchers explore ray 0 in their first iteration, whereas the other half explore ray 1. We require that each strategy in  $\mathcal{S}$  is  $r$ -robust which implies, from the discussion in Section 2, that  $b$  must satisfy  $b^q \leq b_r$ , hence  $b \leq b_r^{1/q}$ . ◀

Note that this upper bound is not only of theoretical significance, but can be obtained in practice via a query-based implementation. This is because the  $i$ -th advice bit can be interpreted, equivalently, as a response to a subset query that asks whether the target is hiding within a specific subset of the infinite line. Informally, the theorem shows which questions to ask to  $k$  different experts<sup>2</sup> about the whereabouts of the target so as to maximize the efficiency of search, while remaining robust to adversarial responses.

We now move to the lower bound. We first show a useful property of parallel search. To illustrate the property, consider Figure 1, which shows the first segments of a  $p$ -parallel strategy defined by  $S_1, \dots, S_p$ . In this example, the first segment of strategies  $S_1, \dots, S_i$  is to the left ray of the line, whereas the first segment of  $S_{i+1}, \dots, S_p$  is to the right ray of the line. Furthermore, the lengths of these segments are in increasing order, in the left and the right ray, respectively, as illustrated. We observe that, without loss of generality, a target that hides infinitesimally beyond the first turnpoint in  $S_j$ , with  $j \in \{1, \dots, p-1\}$  is first discovered by  $S_{j+1}$ , and if  $j = p$ , it is first discovered by  $S_1$ . This is because, if this was not the case, then one of the strategies would mark its second turn before it had explored any new parts of the line, which would mean that the corresponding second segment would be redundant and thus could be omitted.

We can argue, inductively, that the same property extends not only to targets hiding infinitesimally beyond the first turn points, but beyond *every* turn point. To formalize this concept, let  $\mathcal{S}$  be a  $p$ -parallel strategy defined by single-searcher strategies  $S_1, \dots, S_p$ . We say that  $S_j$  is *responsible for the  $i$ -th turn point of  $S_i$*  if  $S_j$  is the first strategy in  $\mathcal{S}$  to discover a

<sup>2</sup> Experts may be inherently erroneous; in Theorem 4 we extend the result to account for query errors.



■ **Figure 1** A snapshot of the first iteration in a  $p$ -parallel strategy.

target hiding infinitesimally beyond the  $i$ -th turnpoint of  $S_i$ . The following lemma shows that it suffices to consider  $p$ -parallel strategies in which a “snapshot” of the first iteration of the  $p$  individual strategies provides a global picture about the relative turnpoints of all individual strategies, for all subsequent iterations. This will help us setup the lower bound.

► **Lemma 2.** *For any  $p$ -searcher strategy  $\mathcal{S}$ , there is a  $p$ -searcher strategy  $\mathcal{S}' = \{S'_1, \dots, S'_p\}$  such that there is a bijection  $\pi : \{1, \dots, p\} \rightarrow \{1, \dots, p\}$  with the property that for any  $j \in \{1, \dots, p\}$ ,  $S'_{\pi(j)}$  is responsible for the  $i$ -th turn point of  $S'_j$  for all  $i \in \mathbb{N}^+$ , and  $\mathcal{S}'$  has competitive ratio no worse than  $\mathcal{S}$ .*

We now show how to prove the lower bound.

► **Theorem 3.** *For any  $r \geq 9$ , every  $r$ -robust strategy for searching on the line with untrusted advice of size  $k$  has consistency at most  $1 + 2^{\frac{k^{1/q}}{r-1}}$ , where  $q = 2^{k-1}$ .*

**Proof sketch.** For convenience of notation, let  $n = 2^k$ , and let  $\mathcal{S}$  be an  $n$ -parallel strategy defined by  $S_1, \dots, S_n$ , and which satisfies the property of Lemma 2. Let  $i$  be such that strategies  $S_1, \dots, S_i$  start their first iteration to the left (in increasing order of this length), whereas  $S_{i+1}, \dots, S_n$  start their first iteration to the right (again in increasing length order). Thus, we have that  $S_j$  is responsible for  $S_{j-1}$ , for all  $j \in \{2, \dots, 2^k\}$ , whereas  $S_1$  is responsible for  $S_n$ . The situation is depicted in Figure 1, where  $p = n$ .

Let  $s_{j,m}$  denote the search length of the  $m$ -th iteration of  $S_j$ . For any fixed  $m$ , consider a target hiding infinitesimally beyond the  $m$ -th turn point of  $S_j$ , for each  $j$ . Since  $S_{j+1}$  is responsible for  $S_j$ , for all  $j \in \{1, \dots, i-1, i+1, \dots, n-1\}$ , we have

$$\text{Cr}(\mathcal{S}) \geq 1 + 2^{\frac{\sum_{l=1}^{m-1} s_{j+1,l}}{s_{j,m}}}, \quad \text{for all } j \in \{1, \dots, i-1, i+1, \dots, n-1\}. \quad (3)$$

In addition, since  $S_{i+1}$  and  $S_1$  are responsible for  $S_i$  and  $S_n$ , respectively, we have that

$$\text{Cr}(\mathcal{S}) \geq 1 + 2^{\frac{\sum_{l=1}^{m-1} s_{i+1,l}}{s_{i,m-1}}} \quad \text{and} \quad \text{Cr}(\mathcal{S}) \geq 1 + 2^{\frac{\sum_{l=1}^{m-1} s_{1,l}}{s_{n,m-1}}}, \quad (4)$$

and note the subtle, but important differences in the indexing of the denominators between (3) and (4). This motivates our next step, in which we *label* the lengths of all search segments of the  $2^k$  strategies in  $\mathcal{S}$  in a way that will allow us to use the above lower bounds. Let  $\{x_l\}_{l=1}^{\infty}$  denote the set of all segment lengths in the parallel strategy  $\mathcal{S}$ . We map bijectively each such length to a segment length of one of the strategies  $S_j$ , according to the following function.

$$s_{j,m} = \begin{cases} x_{j+mn}, & \text{if } j \in [1, \dots, i-1] \\ x_{j-1+mn} & \text{if } j \in [i+1, \dots, n-1] \\ x_{n-1+mn}, & \text{if } j = i \\ x_{n+mn}, & \text{if } j = n. \end{cases} \quad (5)$$

Combining (3), (4) and the mapping (5), we can show that

$$\text{Cr}(\mathcal{S}) \geq 1 + 2 \sup_m \frac{\sum_{l=1}^{mn} x_l}{\sum_{l=0}^{n-1} x_{mn-1+l}}. \quad (6)$$

Define the functional  $F_m$  to be  $F_m = \frac{\sum_{l=1}^{mn} x_l}{\sum_{l=0}^{n-1} x_{mn-1+l}}$ . It is easy to see that this functional satisfies the conditions of Gal's Theorem [26]. Therefore,

$$\text{Cr}(\mathcal{S}) \geq 1 + 2 \sup_m \frac{\sum_{l=1}^{mn} \alpha^l}{\sum_{l=0}^{n-1} \alpha^{mn-1+l}}, \quad \text{where } \alpha = \limsup_{l \rightarrow \infty} x_l^{1/l}. \quad (7)$$

If  $\alpha \leq 1$ , the RHS of (7) is unbounded. If  $\alpha > 1$ , (7) gives

$$\text{Cr}(\mathcal{S}) \geq 1 + 2 \sup_m \frac{\alpha^{mn+1} - \alpha}{(\alpha - 1)\alpha^{mn-1} \frac{\alpha^n - 1}{\alpha - 1}} \geq 1 + 2 \frac{\alpha^2}{\alpha^n - 1}. \quad (8)$$

We will now use the fact that each strategy in  $\mathcal{S}$  is individually  $r$ -robust, in order to bound  $\alpha$  from below, and thus  $\text{Cr}(\mathcal{S})$  as well. From [31] we know that any  $r$ -competitive single searcher strategy of the form  $Y = (y_j)_{j=1}^{\infty}$  satisfies  $y_j = O(b_r^j)$ . Given the labeling scheme (5), it follows that  $x_j = O(b_r^{j/n})$ , hence from the definition of  $\alpha$ ,  $\alpha \leq b_r^{1/n}$ . Therefore, (8) gives

$$\text{Cr}(\mathcal{S}) \geq 1 + 2 \frac{b_r^{2/n}}{b_r - 1} = 1 + 2 \frac{b_r^{1/q}}{b_r - 1},$$

which completes the proof.  $\blacktriangleleft$

We give some intuition behind the significance of the mapping (5) in the proof. The labeling accomplishes two goals: First, it leads to (6), whose sums in the numerator and the denominator contain summands with ‘‘contiguous’’ indices: this is an essential requirement for the application of Gal's theorem. Second, it implies that each strategy  $S_j$  is of the form  $(x_{\pi(j+nl)})_{l=0}^{\infty}$ , where  $\pi$  is a bijection over  $\{1, \dots, n\}$ , which allows us to argue that  $\alpha \leq b_r^{1/n}$ .

### 3.1 Extension to weak predictions

We show how to extend the upper bound to incorporate weak predictions. In this setting, as discussed in Section 1, given advice of size  $k$ , and robustness requirement  $r \geq 9$ , the searcher specifies a tolerance parameter  $H \leq k/2$ . The objective is to obtain an  $r$ -robust strategy of minimum competitive ratio assuming that at most  $H$  advice bits are erroneous.

To address this problem, we will make use of a result by Rivest *et al.* [50], who studied games with a *lying responder*. In their setting, given  $k \in \mathbb{N}^+$ ,  $H \leq k/2$ , and a domain  $\mathcal{D} = \{1, \dots, m\}$ , the objective is to find the index of an unknown  $x \in \mathcal{D}$ , using  $k$  queries, of which up to  $H$  may receive incorrect responses. A query can be a comparison query of the form ‘‘Is  $x \leq M$ ?’’, for some given  $M \in [1, m]$ , or more generally, a subset query of the form ‘‘Is  $x$  in  $S$ ?’’, where  $S$  is a subset of the domain  $\mathcal{D}$ . Define the sum of binomial coefficients  $\binom{N}{m} := \sum_{j=0}^m \binom{N}{j}$ , for  $m \leq N$ . In [50] it was shown that as long as  $m \leq 2^k / \binom{k}{H}$ ,  $k$  comparison queries suffice to find  $x$  in the above game, in the presence of at most  $H \leq k/2$  errors. This leads to the following extension of Theorem 1.

**► Theorem 4.** *For any  $r \geq 9$ , and any  $H \leq k/2$ , there is an  $r$ -robust strategy for searching on the line with  $k$ -bit advice that has competitive ratio at most  $1 + 2 \frac{b_r^{1/q}}{b_r - 1}$ , where  $q = 2^{k-1} / \binom{k}{H}$ .*

We can further show that

$$\frac{1}{q} \leq \frac{1}{2^{k(1-\mathcal{H}(\frac{H}{k}))}-1},$$

where  $\mathcal{H}$  denotes the *binary entropy* function. This allows for a direct comparison to the Pareto-optimal upper bound of Theorem 1. In particular, we observe that as  $k$  increases, the effect of the advice error in the competitive ratio becomes marginal, even if  $H$  is as high as linear in  $k$ .

#### 4 Ray search with directional prediction

In this section, we study  $m$ -ray search in the setting in which the prediction is the ray of the hiding target. Without loss of generality, we suppose that the prediction is the ray indexed 0.

For the upper bound, we consider the following strategy that generalizes the biased search approach of [20]. The searcher fixes some  $b > 1$  and  $\delta > 1$ , to be specified later, and explores the rays in the cyclic order  $0, 1, \dots, m-1$ . If the ray visited in the  $i$ -th iteration is ray 0, it explores it to a length equal to  $\delta b^i$ , otherwise, i.e., if the visited ray is in  $\{1, \dots, m-1\}$ , it explores it to a length equal to  $b^i$ . Thus the search combines elements of geometric search with a bias towards the predicted ray, as expressed by the parameter  $\delta$ .

► **Theorem 5.** *For every  $b > 1$ ,  $\delta > 1$ , the strategy described above has consistency at most  $1 + 2\frac{b^m}{b^m-1} + \frac{2}{\delta}\frac{b^m}{b^m-1}\frac{b^m-b}{b-1}$ , and robustness at most  $1 + 2\delta\frac{b^{m+1}}{b^m-1} + 2\frac{b^{m+1}}{b^m-1}\frac{b^m-b}{b-1} - 2b^m$ .*

Observe that if  $\delta = 1$ , then both the consistency and the robustness of the above strategy are equal to  $1 + 2\frac{b^m}{b^m-1}$ , as expected (i.e., the competitive ratio of a geometric strategy with base  $b$ ). For any fixed  $b$ , by increasing  $\delta$ , the consistency of the resulting strategy improves, at the expense of its robustness. We would like thus to optimize the robustness by choosing  $\delta$  as a function of  $b$ ,  $m$  and the desired consistency, however it is not obvious that this is possible analytically. Instead, suppose that we choose  $b = (m+1)/m$ , namely the base of the geometric strategy that results in an optimal competitive ratio for  $m$ -ray search equal to  $r_m^* = 1 + 2\rho_m^*$ . Then  $\frac{b^m}{b^m-1} \leq e/(e-1)$ , which implies that the consistency of the strategy is at most

$$1 + 2\frac{e}{e-1} + \frac{2}{\delta}\frac{e}{e-1}(\rho_m^* - m) = \frac{2}{\delta}\frac{e}{e-1}(\rho_m^* - m) + O(1).$$

On the other hand, the robustness of the strategy is at most

$$1 + 2\delta\frac{81}{36} + 2\frac{81}{36}(\rho_m^* - m) - 4 = \frac{9}{2}(\delta + \rho_m^* - m) + O(1).$$

Therefore, if we would like the strategy to be  $c$ -consistent, where  $c = O(1) + 2\tilde{c}$ , for some  $\tilde{c}$ , we can choose  $\delta$  to be equal to  $\frac{e}{(e-1)\tilde{c}}(\rho_m^* - m)$ , and the resulting robustness is then at most  $\frac{9}{2}(\rho_m^* - m)(1 + \frac{e}{(e-1)\tilde{c}}) + O(1)$ . We can also obtain more precise tradeoffs as  $m \rightarrow \infty$ , since in this case, it is known that  $r_m^* = 1 + 2\rho_m^* = 1 + 2em$ .

► **Corollary 6.** *For  $m \rightarrow \infty$ , the above strategy has consistency at most  $1 + 2\frac{e}{e-1} + \frac{2}{\delta}em$ , and robustness at most  $2e(m + \frac{\delta}{e-1})$ . In particular, given  $\tilde{c}$ , the strategy is  $(O(1) + 2\tilde{c})$ -consistent, and  $(O(1) + 2em(1 + \frac{e}{(e-1)\tilde{c}}))$ -robust.*

Next, we show a negative result on the tradeoff between the consistency and robustness that any strategy can achieve. The proof follows an approach that we generalize in the weak predictions setting (proof of the lower bound in Theorem 8).

► **Theorem 7.** *Any  $c$ -consistent strategy for searching with directional prediction, where  $c = 1 + 2\tilde{c}$ , has robustness at least  $1 + 2\rho_{m-1}^*(1 + \frac{1}{\tilde{c}-1})$ . In particular, for  $m \rightarrow \infty$ , its robustness is at least  $1 + 2e(m-1)(1 + \frac{1}{\tilde{c}-1})$ .*

#### 4.1 Extension to weak predictions

We consider the model in which if the predicted ray is  $h \in \{0, \dots, m-1\}$ , then the searcher would like to minimize its competitive ratio, assuming that the target hides in one of the rays in the interval  $[h - H \bmod m, h + H \bmod m]$ , where  $H \leq m/2$  is the tolerance parameter. This captures the case that the hiding ray is expected to be in a vicinity of the predicted ray, with respect to the tolerance of the searcher. We denote this set of rays by  $R_H$ , and its complement, by  $\bar{R}_H$ , and note that  $|R_H| = 2H + 1$  and  $|\bar{R}_H| = m - 2H - 1$ . Without loss of generality, we may assume that  $R_H = \{0, \dots, 2H\}$ .

We prove an asymptotically tight bound on the tradeoff between the competitive ratio and the robustness, that generalizes the error-free setting. Note that every strategy has competitive ratio at least  $1 + 2\rho_{2H+1}^*$ , since the weak prediction may incur a search in a  $(2H+1)$ -ray star. We also obtain an interesting corollary: as the competitive ratio approaches the optimal bound of  $1 + 2\rho_{2H+1}^*$ , the robustness increases dramatically.

► **Theorem 8.** *For every  $\tilde{c} > \rho_{2H-1}^*$  there exists a strategy with directional hint that has competitive ratio  $1 + 2\tilde{c}$ , and robustness at most  $O(\frac{\rho_{2H+1}^*}{\tilde{c} - \rho_{m-2H-1}^*}(m - 2H))$ . Furthermore, this bound is tight, i.e., every strategy of competitive ratio  $1 + 2\tilde{c}$  has robustness at least  $\Omega(\frac{\rho_{2H+1}^*}{\tilde{c} - \rho_{m-2H-1}^*}(m - 2H))$ .*

**Proof.** We first prove the upper bound, which generalizes the strategy we used in the context of consistency/robustness tradeoffs. Define  $b = \frac{2H+1}{2H}$ , i.e., the optimal base of a geometric strategy for searching in a  $(2H+1)$ -ray star, and  $\delta > 1$ , to be specified later. Consider a cyclic strategy which visits rays  $0, \dots, m-1$  in this order, and which works in rounds. Specifically, in round  $i$ , it explores ray  $j \in R_H$  to length  $\delta b^{(2H+1)i+j}$ , and every ray in  $\bar{R}_H$  to length  $b^{(2H+1)i+2H}$ . The competitive ratio of this strategy, assuming error at most  $H$ , is maximized for targets hiding infinitesimally beyond the turn points on ray  $2H$  in  $R_H$ . Simple calculations show that the competitive ratio is

$$1 + 2\rho_{2H+1}^* + \Theta\left(\frac{1}{\delta} \frac{b^{2H+1}}{b^{2H+1} - 1}(m - 2H - 1)\right),$$

and note that  $\frac{b^{2H+1}}{b^{2H+1} - 1}$  is at most  $e/(e-1)$ , by the choice of  $b$ . Thus, for the competitive ratio to be at most  $1 + 2\tilde{c}$ , it must be that

$$\delta \in \Omega\left(\frac{m - 2H}{\tilde{c} - \rho_{2H+1}^*}\right). \tag{9}$$

The robustness of the strategy is evaluated for a target hiding at distance infinitesimally beyond the turn points of the searcher on ray  $m-1$ . After simple calculations we obtain that the robustness is at most  $O(\delta \rho_{2H+1}^*(m - 2H + 1))$ , which from (9) is at most  $O(\frac{\rho_{2H+1}^*}{\tilde{c} - \rho_{2H+1}^*}(m - 2H))$ , and which proves the upper bound.

We now proceed with the lower bound. Any strategy for the problem consists of *phases*, which alternate between searching a subset of  $R_H$  and a subset of  $\bar{R}_H$ . Namely, every strategy  $X$  is of the form  $X = (x_i)_{i \geq 0}$ , in which  $x_i$ , for even  $i$ , describes the aggregate explored length of  $X$  on rays that belong exclusively in  $R_H$ , and  $x_i$  for  $i$  odd, describes the aggregate explored length on rays that belong exclusively in  $\bar{R}_H$ . Thus, in each phase  $i$ , the searcher incurs a cost of  $2x_i$ , for all  $i$  (except for the phase at which the target is found).

Define  $S_i = \sum_{j=0}^i x_{2j}$ , and  $S'_i = \sum_{j=0}^{i-1} x_{2j+1}$ . Then from the competitiveness of the strategy, for all even  $i$ , and for any target  $t_H$  hiding in  $R_H$  that is discovered in phase  $2(i+1)$ ,

$$1 + 2 \frac{S_i + S'_i}{d(t_H)} \leq c \Rightarrow \tilde{c} \geq \frac{S_i + S'_i}{d(t_H)}. \quad (10)$$

We know that searching in a  $(2H+1)$ -ray has competitive ratio at least  $1 + 2\rho_{2H+1}^*$ . This means that for all  $i$ , there exists a target  $d_i$  that is discovered in phase  $2(i+1)$  such that

$$1 + 2 \sup_i \frac{S_i}{d_i} \geq 1 + 2\rho_{2H+1}^*,$$

hence there exists some  $\bar{i}$  for which the above inequality gives  $1 + 2 \frac{S_{\bar{i}}}{d_{\bar{i}}} \geq 1 + 2\rho_{2H+1}^* - \epsilon$ , where  $\epsilon \rightarrow 0$ , as  $\bar{i}$  is allowed to be unbounded. To simplify the exposition, we can thus assume that  $\epsilon = 0$ , and obtain

$$d(t_{\bar{i}}) \leq \frac{S_{\bar{i}}}{\rho_{2H+1}^*}. \quad (11)$$

Moreover, for any  $i$ , there exists a hiding position for a target  $t'_i$  in  $\bar{R}_H$  that is first discovered in phase  $2i+1$  it must be that

$$1 + 2 \frac{S'_i}{d(t'_i)} \geq m - 2H - 1 \Rightarrow d(t'_i) = O\left(\frac{S'_i}{m - 2H - 1}\right), \quad (12)$$

since  $2S'_i$  describes the cost incurred by the searcher on rays in  $\bar{R}_H$ , right before phase  $2i+1$  starts. Note also that this inequality holds *for all*  $i$ , unlike (11), that holds only for  $\bar{i}$ .

To bound the robustness, consider the phase  $2\bar{i}+1$ , with  $\bar{i}$  as defined above, and the target  $t'_{\bar{i}}$ , again as defined above. Then we have that

$$\text{Robustness} \geq 1 + 2 \frac{S_{\bar{i}} + S'_{\bar{i}}}{d(t'_{\bar{i}})} = \Omega\left(\left(1 + \frac{S_{\bar{i}}}{S'_{\bar{i}}}\right)(m - 2H)\right), \quad (13)$$

from (12). Moreover, from (10) and (11) we have that

$$\frac{S_{\bar{i}} + S'_{\bar{i}}}{S'_{\bar{i}}} \leq \frac{\tilde{c}}{\rho_{2H+1}^*} \Rightarrow \frac{S_{\bar{i}}}{S'_{\bar{i}}} \geq \frac{\rho_{2H+1}^*}{\tilde{c} - \rho_{2H+1}^*},$$

and substituting the above inequality to (10) yields the result.  $\blacktriangleleft$

## 5 Ray search with positional prediction

In this section we study  $m$ -ray searching in the setting in which the prediction is the position of the target in the star environment. Namely, the prediction  $h$  is a pair  $(d_h, u_h)$ , where  $d_h$  is the predicted distance from  $O$  and  $u_h$  is the predicted ray. We first show the upper bound.

► **Theorem 9.** *For any  $r \geq r_m^*$ , there is an  $r$ -robust strategy of consistency at most  $1 + 2 \frac{1}{b_r - 1}$ .*

We will now show that the strategy of Theorem 9 is Pareto-optimal. The proof of the following theorem generalizes, but also simplifies the lower bound of [3] which applies only to linear search ( $m = 2$ ). The crux in the proof is to exploit the properties of the parameter  $\alpha_Y$ , where  $Y$  will be defined as the sequence of search lengths of a cyclic strategy defined by a linear recurrence relation. In particular, these properties allow us to bypass technical complications related to the study of such relations, by establishing appropriate lower bounds (as opposed to solving the recurrence relation).

## 12:12 Competitive Search in the Line and the Star with Predictions

► **Theorem 10.** For any  $r \geq r_m^*$ , no  $r$ -robust strategy for searching with positional prediction has consistency better than  $1 + 2\frac{1}{b_r - 1}$ .

**Proof sketch.** Using techniques rooted in previous studies of star search, we can show that there exists a *cyclic* strategy of the form

$$y_{j_h} = h, \text{ for some index } j_h \quad \text{and} \quad 1 + 2\frac{\sum_{i=0}^j y_i}{y_{j-m+1}} = r, \text{ for all } j \in [m-1, j_h],$$

which is  $r$ -robust and minimizes the consistency (among all  $r$ -robust strategies). It follows that for all  $j \leq j_h$ , the search length  $y_j$  is determined by the recurrence

$$y_j = r(y_{j-m+1} - y_{j-m}), \quad (14)$$

with some initial conditions  $y_0, \dots, y_{m-1}$ . We have that

$$d(Y, h) = \sum_{j=0}^{j_h} y_j = d_h + 2 \sum_{j=0}^{j_h-1} y_j = d_h + 2r(y_{j_h-m+1} - y_0) + \sum_{j=0}^{m-1} y_j, \quad (15)$$

where we used the fact that  $\sum_j y_j$  is telescoping, as seen by (14). From (15) we have  $\frac{d(Y, h)}{d_h} = 1 + 2\frac{ry_{j_h-m+1}}{d_h} + \frac{1}{d_h}(\sum_{j=0}^{m-1} y_j - y_0)$ . Since  $Y$  is cyclic, and  $d_h = y_{j_h}$ , we obtain that

$$\sup_h \frac{d(Y, h)}{d_h} \geq 1 + 2 \sup_{j_h} \frac{ry_{j_h-m+1}}{y_{j_h}} = 1 + 2r \sup_{j_h} \frac{y_{j_h-m+1}}{y_{j_h}}. \quad (16)$$

Define the functional  $F_j(Y) = \frac{y_{j-m+1}}{y_j}$ . This functional satisfies the conditions of Gal's functional theorem [26], hence  $\sup_j \frac{y_{j-m+1}}{y_j} \geq \frac{\alpha_Y^{j-m+1}}{\alpha_Y} = \alpha_Y^{-m}$ . Since  $Y$  is  $r$ -robust, as discussed in Section 2  $r \geq \text{Cr}(X) \geq \frac{\alpha_Y^m}{\alpha_Y - 1}$ , where it must be  $\alpha_Y \leq b_r$ , from the definition of  $b_r$ . Thus, (16) gives

$$\sup_h \frac{d(Y, h)}{d_h} \geq 1 + 2\frac{\alpha_Y^m}{\alpha_Y - 1}\alpha_Y^{-m} = 1 + 2\frac{1}{\alpha_Y - 1} \geq 1 + 2\frac{1}{b_r - 1}.$$

Hence, the consistency of  $Y$  is at least  $1 + 2\frac{1}{b_r - 1}$ , and thus so is the consistency of  $X$ . ◀

### 5.1 Extension to weak predictions

Given the prediction  $h$  related to a target  $t$ , we define the prediction *error* as the distance between  $t$  and  $h$  in the star, normalized by the distance  $d(h)$  of the prediction's position from the origin. Namely,  $\eta = \frac{|d(t) - d(h)|}{d(h)}$ . We distinguish between different types of the error: If  $d$  and  $h$  are in the same ray, but  $d(t) > d(h)$  we call the error *positive*, whereas if  $d$  and  $h$  are in the same ray, but  $d(t) < d(h)$  we call the error *negative*. If the error is neither positive or negative, then  $h$  and  $t$  are in different rays.

Let  $X_h$  denote the Pareto-optimal strategy of Theorem 9. Let  $H > 0$  denote the tolerance parameter that is specified by the searcher, and consider the strategy  $X_{h(1+H)}$ , i.e., the strategy that pretends that the prediction is at the same ray as  $h$ , but at distance  $d(h)(1+H)$  from  $O$ . The following result is a corollary of Theorem 9.

► **Corollary 11.** For any  $H > 0$  and  $r \geq r_m^*$ , strategy  $X_{h(1+H)}$  is  $r$ -robust and has competitive ratio at most  $\min\{1 + 2\frac{1+H}{b_r - 1}, r\}$ , if the error is either positive or negative, and at most  $H$ . Otherwise, its competitive ratio is at most  $r$ .

Last, we can show that the above tradeoffs are tight.

► **Theorem 12.** For any  $r$ -robust strategy with positional prediction, there exists  $q > 0$  such that its competitive ratio is no better than  $\min\{1 + 2\frac{1+q}{b_r - 1}, r\}$  for positive or negative error at most  $q$ . Otherwise, its competitive ratio is at least  $r$ .

---

References

---

- 1 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, 2003.
- 2 Steve Alpern and Thomas Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2):265–279, 2013.
- 3 Spyros Angelopoulos. Online search with a hint. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 51:1–51:16, 2021.
- 4 Spyros Angelopoulos. Further connections between contract-scheduling and ray-searching problems. *Journal of Scheduling*, 25(2):139–155, 2022.
- 5 Spyros Angelopoulos, Christoph Dürr, and Shendan Jin. Best-of-two-worlds analysis of online search. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 7:1–7:17, 2019.
- 6 Spyros Angelopoulos, Christoph Dürr, and Thomas Lidbetter. The expanding search ratio of a graph. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 9:1–9:14, 2016.
- 7 Spyros Angelopoulos and Shahin Kamali. Contract scheduling with predictions. *Journal of Artificial Intelligence Research*, 77:395–426, 2023.
- 8 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory G.E. Rawlins. Searching in the plane. *Information and Computation*, 106:234–244, 1993.
- 9 Siddhartha Banerjee, Vincent Cohen-Addad, Anupam Gupta, and Zhouzi Li. Graph searching with predictions. In *Proceedings of the 14th Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 251 of *LIPICs*, pages 12:1–12:24, 2023.
- 10 Anatole Beck. On the linear search problem. *Naval Research Logistics*, 2:221–228, 1964.
- 11 Anatole Beck and Donald J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8:419–429, 1970.
- 12 Richard Bellman. An optimal search problem. *SIAM Review*, 5:274, 1963.
- 13 Daniel S. Bernstein, Lev Finkelstein, and Shlomo Zilberstein. Contract algorithms and robots on rays: Unifying two scheduling problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1211–1217, 2003.
- 14 Lucas Boczkowski, Amos Korman, and Yoav Rodeh. Searching a tree with permanently noisy advice. In *ESA 2018-26th Annual European Symposium on Algorithms*, pages 1–32, 2018.
- 15 Anthony Bonato, Konstantinos Georgiou, Calum MacRury, and Pawel Pralat. Probabilistically faulty searching on a half-line – (Extended abstract). In *LATIN 2020: Theoretical Informatics – 14th Latin American Symposium, São Paulo, Brazils*, volume 12118 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2020.
- 16 Prosenjit Bose, Jean-Lou De Carufel, and Stephane Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.
- 17 Marek Chrobak, Leszek Gasieniec, Thomas Gorry, and Russell Martin. Group search on the line. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 164–176. Springer, 2015.
- 18 Anne Condon, Amol Deshpande, Lisa Hellerstein, and Ning Wu. Algorithms for distributional and adversarial pipelined filter ordering problems. *ACM Transaction on Algorithms*, 5(2):24:1–24:34, 2009.
- 19 Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, and Jaroslav Opatrný. Search on a line with faulty robots. *Distributed Comput.*, 32(6):493–504, 2019.
- 20 Erik D Demaine, Sándor P Fekete, and Shmuel Gal. Online searching with turn cost. *Theoretical Computer Science*, 361:342–355, 2006.
- 21 Stefan Dobrev, Rastislav Kráľovič, and Euripides Markou. Online graph exploration with advice. In *Structural Information and Communication Complexity: 19th International Colloquium, SIROCCO 2012*, pages 267–278. Springer, 2012.

- 22 Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(9), pages 9732–9740, 2022.
- 23 Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Information and Computation*, 206(11):1276–1287, 2008.
- 24 Shmuel Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.
- 25 Shmuel Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27:17–30, 1974.
- 26 Shmuel Gal. *Search Games*. Academic Press, 1980.
- 27 Subir Kumar Ghosh and Rolf Klein. Online algorithms for searching and exploration in the plane. *Comput. Sci. Rev.*, 4(4):189–201, 2010.
- 28 Barun Gorain and Andrzej Pelc. Deterministic graph exploration with advice. *ACM Transactions on Algorithms*, 15(1):1–17, 2018.
- 29 C. Hipke, C. Icking, R. Klein, and E. Langetepe. How to find a point in the line within a fixed distance. *Discrete Applied Mathematics*, 93:67–73, 1999.
- 30 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2733–2743, 2021.
- 31 Parick Jaillet and Matthew Stafford. Online searching. *Operations Research*, 49:234–244, 1993.
- 32 Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Yin. Optimal constructions of hybrid algorithms. *Journal of Algorithms*, 29(1):142–164, 1998.
- 33 Ming-Yang Kao, John H Reif, and Stephen R Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–80, 1996.
- 34 David G. Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2009.
- 35 Dennis Komm, Rastislav Kráľovič, Richard Kráľovič, and Jasmin Smula. Treasure hunt with advice. In *Structural Information and Communication Complexity: 22nd International Colloquium, SIROCCO 2015*, pages 328–341. Springer, 2015.
- 36 E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proc. of the 23rd Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 280–289, 1996.
- 37 Andrey Kupavskii and Emo Welzl. Lower bounds for searching robots, some faulty. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 447–453, 2018.
- 38 Elmar Langetepe. On the optimality of spiral search. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–12. SIAM, 2010.
- 39 Russell Lee, Jessica Maghakian, Mohammad H. Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. Online peak-aware energy scheduling with untrusted advice. In *Proceedings of the 12th ACM International Conference on Future Energy Systems (eEnergy)*, pages 107–123. ACM, 2021.
- 40 Tongxin Li, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven H. Low. Robustness and consistency in linear quadratic control with untrusted predictions. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1):18:1–18:35, 2022.
- 41 Thomas Lidbetter. Search and rescue in the face of uncertain threats. *Eur. J. Oper. Res.*, 285(3):1153–1160, 2020.
- 42 Alexander Lindermayr and Nicole Megow. Repository of works on algorithms with predictions. <https://algorithms-with-predictions.github.io/about>, 2023. Accessed: 2023-04-01.
- 43 Alejandro López-Ortiz and Svem Schuierer. The ultimate strategy to search on  $m$  rays? *Theoretical Computer Science*, 261(2):267–295, 2001.

- 44 Alejandro López-Ortiz and Sven Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theoretical Computer Science*, 310(1–3):527–537, 2004.
- 45 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021.
- 46 Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. The oil searching problem. In *Proceedings of the 17th European Symposium on Algorithms (ESA)*, pages 504–515, 2009.
- 47 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.
- 48 Andrzej Pelc and Ram Narayan Yadav. Advice complexity of treasure hunt in geometric terrains. *Information and Computation*, 281:104705, 2021.
- 49 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 9661–9670, 2018.
- 50 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *J. Comput. Syst. Sci.*, 20(3):396–404, 1980.
- 51 Sven Schuierer. Lower bounds in online geometric searching. *Computational Geometry: Theory and Applications*, 18(1):37–53, 2001.
- 52 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2020.



# Rényi-Ulam Games and Online Computation with Imperfect Advice

Spyros Angelopoulos  

CNRS and LIP6-Sorbonne University, Paris, France

Shahin Kamali  

York University, Toronto, Canada

---

## Abstract

We study the nascent setting of online computation with *imperfect* advice, in which the online algorithm is enhanced by some prediction encoded in the form of an imperfect, and possibly erroneous binary string. The algorithm is oblivious to the advice error, but defines a desired *tolerance*, namely an upper bound on the number of erroneous advice bits it can tolerate. This is a model that generalizes the *Pareto-based* advice model, in which the performance of the algorithm is only evaluated at the extreme values of error (namely, if the advice has either no errors, or if it is generated adversarially). It also subsumes the model in which the algorithm elicits a prediction on the online sequence, via imperfect responses to a number of *binary queries*.

In this work, we establish connections between games with a lying responder, also known as *Rényi-Ulam games*, and the design and analysis of online algorithms with imperfect advice. Specifically, we demonstrate how to obtain upper and lower bounds on the competitive ratio for important online problems such as time-series search, online bidding, and fractional knapsack. Our techniques provide the first lower bounds for online problems in this model. We also highlight and exploit connections between competitive analysis with imperfect advice and fault-tolerance in multiprocessor systems. Last, we show how to waive the dependence on the tolerance parameter, by means of resource augmentation and robustification.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online computation, Rényi-Ulam games, query models, beyond worst-case analysis

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.13

**Related Version** *Full Version:* [arxiv.org/abs/2301.01631](https://arxiv.org/abs/2301.01631)

**Funding** The first author was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR). The second author was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number DGECR-2018-00059].

## 1 Introduction

Online computation, and *competitive analysis*, in particular, have served as the definitive framework for the theoretical analysis of algorithms in a state of uncertainty. While the early, standard definition of online computation [37] assumes that the algorithm has no knowledge in regard to the request sequence, in practical situations, the algorithm may indeed have certain limited, but possibly inaccurate such information (e.g., some lookahead, or historical information on typical sequences). Hence, there is a clear need for more nuanced models that capture the power and limitations of online algorithms enhanced with external information.

One such approach, within Theoretical Computer Science, is the framework of *advice complexity*; see [18, 9, 20], the survey [10] and the book [25]. In the advice-complexity model (and in particular, the *tape* model [8, 9]), the online algorithm receives a string that encodes information concerning the request sequence, and which can help improve its performance. The objective is to quantify the tradeoffs between the size of the advice (in terms of the



© Spyros Angelopoulos and Shahin Kamali;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics



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

number of bits) and the competitive ratio of the algorithm. This model places stringent requirements: the advice is assumed to be error-free, and may be provided by an omnipotent oracle. Thus, as noted in [34], this model is mostly of theoretical significance.

A different and more practical approach studies the effect of *predictions* towards improving the competitive ratio. In this model, the online algorithm is enhanced with some imperfect information concerning the request sequence, without restrictions on its size. One is interested in algorithms whose performance degrades gently as a function of the prediction *error*, and specifically perform well if the prediction is error-free (what is called the *consistency* of the algorithm), but also remain robust under any possible error (what is called the *robustness* of the algorithm). This line of research was initiated with the works [31] and [35], and a large number of online problems have been studied under this model (see, e.g., the survey [34] and the online collection [29]).

A combination of the advice complexity and prediction models is the *untrusted or Pareto-based advice* model, introduced in [5]. Here, parts of the advice may be erroneous, and the algorithm's performance is evaluated in two extreme situations, in regard to the advice error. At the one extreme, the advice is error-free, whereas, at the other extreme, the advice is generated by a (malicious) adversary who aims to maximize the performance degradation of the algorithm. Using the terminology of algorithms with predictions, these two competitive ratios are called consistency and robustness, respectively. The objective is to identify algorithms that are *Pareto-efficient*, and ideally *Pareto-optimal*, i.e., attain the best-possible tradeoffs between these two extreme measures. Several online problems have been studied recently within this framework of Pareto-optimality (both within the advice and the predictions models); see, e.g., [39, 28, 26, 4, 6].

## 1.1 Online computation with imperfect advice

The starting observation that motivates this work is that the Pareto-based framework of untrusted advice only focuses on extreme competitive ratios, namely the consistency and the robustness. A more general issue, instead, is to evaluate the impact of the advice error on the performance of the online algorithm. Given an advice string of size  $k$ , let us denote by  $\eta \leq k$  the number of erroneous bits. Naturally, the algorithm does not know the exact advice error ahead of time. Instead, the algorithm defines an application-specific parameter  $H \leq k$  which determines the desired *tolerance* to errors, or, equivalently, an anticipated upper bound on the advice error. This is motivated by recent works in learning-enhanced online algorithms with *weak predictions*, in which the prediction is an upper bound of some pertinent parameter of the input (see e.g., online knapsack with frequency predictions [23], where the prediction is an upper bound on the number of items of each value that appear online). Our objective is to quantify the tradeoffs between advice size, tolerance and competitive ratio, both from the point of upper and lower bounds.

A different interpretation of imperfect advice treats each advice bit as a (potentially erroneous) response to a *binary query* concerning the input. Hence, one may think of  $k$ -bit advice as a prediction elicited by means of  $k$  imperfect binary experts. Note that queries are known to help improve the performance of approximation algorithms in ML applications. For example, [33] studied clustering with noisy queries, where a query asks whether two points should belong in the same cluster, and where each query receives a correct response with probability  $p$  that is known to the algorithm. A different example is parsimonious learning-augmented caching [22], in which the system learns the predicted next-arrival time of certain appropriately queried pages.

In this work, we study the power, but also the limitations of online algorithms with adversarially erroneous queries. Unlike [33], we do not rely on any probabilistic assumptions concerning the query responses. To our knowledge, the imperfect advice model (in particular, its binary query-based interpretation) has only been applied to the problems of *contract scheduling* [6] and time-series search [7], from the point of view of upper bounds. While these works showed that binary queries help improve the algorithmic performance, both in terms of theoretical and empirical analysis, no principled methodology for obtaining lower bounds has been developed so far.

## 1.2 Contribution

We establish connections between games with a lying responder and the design and analysis of online algorithms with imperfect advice. Namely, we show how to leverage results from the analysis of *Rényi-Ulam* games, and obtain both positive and negative results on the competitive analysis. We apply these tools to three important and well-studied online problems, namely time-series search, online bidding, and online fractional knapsack. Our results improve the known upper bounds for these problems, where such results were already known, but also provide the first lower bounds on the competitive ratio of online problems in this setting, without any restrictive assumptions.

More precisely, we begin as a warm-up<sup>1</sup> with the time-series search problem in Section 3, which illustrates how these techniques can help us improve upon the results of [7]; we also show how to evaluate the competitive ratios, using approximations based on the binary entropy function. In Section 4, we study a more complex application, namely the online bidding problem, first studied in [5] in the context of untrusted advice. Here, the crucial part is establishing near-optimal lower bounds. We achieve this by formulating a multi-processor version of online bidding in  $l \leq 2^k$  processors, in which a certain number of processors may be faulty; we then relate the competitive ratio of this problem to the imperfect advice setting, by relating fault-tolerance in the processor level, to the inherent error in Rényi-Ulam games. In Section 5 we study the online fractional knapsack problem. Here, we present an algorithm whose competitive ratio converges to 1 at a rate exponential in  $k$ , as long as  $H < k/2$ . We also present a near-matching lower bound that shows that our algorithm is close-to-optimal. For the upper bound, the crux is to allocate queries so as to approximate two appropriately defined parameters of the instance. For the lower bound, we use an information theoretic argument. Specifically, we show a reduction from Rényi-Ulam games: if there existed an algorithm of competitive ratio better than a certain value, one could play the game beyond the theoretical performance bound, which is a contradiction.

As explained above, the parameter  $H$  expresses the algorithm’s desired tolerance to errors, and is thus application-specific. In Section 6 we show how to waive the assumption that the precise tolerance is known ahead of time, in two different ways: First, by *resource-augmentation* arguments, i.e., by comparing the performance of an algorithm with perfect (error-free) advice of size  $k$  to that of an algorithm with  $l > k$  advice bits but potentially very high advice error. Second, by *robustifying* the algorithm, namely by requiring that the algorithm performs well even if the error happens to exceed the tolerance parameter.

The techniques we develop can be applicable to other online problems. Specifically, our approach to the online bidding problem defines the following general framework: For upper bounds, one would aim to define a collection of “candidate” algorithms that are closely

---

<sup>1</sup> For *ski rental*, which is another canonical warm-up problem, [5] showed that a single advice bit suffices to obtain optimal consistency/robustness. Hence, the problem is resolved under the imperfect advice model as well.

ranked in terms of their worst-case performance. Then the advice can be used so as to select a suitable candidate from this collection that is close to the best-possible. For lower bounds, one would aim to show that in any collection of candidate algorithms, the erroneous queries may have to always return a solution sufficiently far, in terms of “rank”, from the best one; then one needs to relate the concept of “rank” to performance, from a lower-bound point of view. This last part highlights connections between an online problem with adversarial advice and its fault-tolerant version in a parallel system (with no advice). On the other hand, our approach to the time-series and fractional knapsack problems illustrate another general technique: For upper bounds, one should identify some important parameters of the problem, then allocate the queries appropriately so as to approximate them in the presence of response errors. For lower bounds, information-theoretic arguments should establish a reduction from a Rényi-Ulam game to the online problem.

There are two additional observations concerning the results in this work. First, we allow *adaptive* queries, in that the response to the  $i$ -th query is a function of responses to the previous  $i - 1$  queries. Second, it is important to note that the results we present cannot be obtained straightforwardly by applying some error-correcting code. More precisely, one may be tempted to dedicate some advice bits towards error correction and use the remaining error-free bits in the spirit of classic advice complexity results. However, such an approach may very well be suboptimal since, depending on the problem at hand, an optimal algorithm may benefit more from a large number of somewhat erroneous advice bits than from a smaller number of perfect bits, and the analysis must take into account this possibility.

Due to space limitations, we omit or only sketch certain technical proofs. We refer to the full version on arXiv for the complete proofs.

## 2 Games with a lying responder

We review some core results related to games with a lying responder which will be in the heart of the analysis of online problems with imperfect advice. We are particularly interested in [36], which studied games between a *questioner* and a *responder*, related to an unknown value  $x$  drawn from a domain  $\mathcal{D}$ , where  $\mathcal{D}$  is a subset of reals or in general a totally ordered set. The questioner may ask general queries of the form “is  $x$  in  $S$ ”, where  $S$  is some subset of  $\mathcal{D}$ , and which are called *subset* queries. The upper bounds of [36] hold even if the questioner asks much simpler queries, namely *comparison* queries of the form “is  $x$  at most  $a$ ”, for some given  $a$ . Both the upper and lower bounds in [36] are expressed in terms of partial sums of binomial coefficients. Formally, we define:

$$\binom{N}{m} := \sum_{j=0}^m \binom{N}{j}, \text{ for } m \leq N.$$

We are interested, in particular, in the following game played over a continuous space:

**CONTINUOUSSEARCH( $k, H$ ) game.** In this game,  $x$  is a real number with  $x \in \mathcal{D} = (0, 1]$ , and the questioner asks  $k$  queries, at most  $H$  of which may receive erroneous responses. The objective of the questioner is to find an interval  $I_x$  such that  $x \in I_x$  and  $|I_x|$  is minimized.

► **Lemma 1** ([36]). *Any questioner’s strategy for CONTINUOUSSEARCH( $k, H$ ) with  $H \leq k/2$  is such that  $|I_x| \geq \binom{k}{H} / 2^k$ . Moreover, for  $H \leq k/2$ , there is a strategy, named C-WEIGHTING, that uses comparison queries and outputs an interval  $I_{W,x}$  with  $|I_{W,x}| \leq \binom{k-H}{H} / 2^{k-H}$ .*

The following game will be useful in our analysis of online time-series and fractional knapsack.

**FIND( $k, H$ ) game.** In this game, given  $k$  and  $H \leq k/2$ , and  $\mathcal{D} = \{1, \dots, m\}$ , the objective is to find an unknown  $x \in \mathcal{D}$ , using  $k$  queries, up to  $H$  of which may be answered incorrectly.

The proof of the following theorem is direct from Lemma 1:

► **Theorem 2.** *The largest positive integer  $\mu(k, H)$  such that a questioner can identify any number  $x \in \{1, 2, \dots, \mu(k, H)\}$  in the FIND( $k, H$ ) game is such that*

$$2^{k-H} / \binom{k-H}{H} \leq \mu(k, H) \leq 2^k / \binom{k}{H}.$$

We define two further games that will be of interest to our analysis. The first is related to searching in cyclic permutations, and will be useful in the upper-bound analysis of online bidding.

**MINCYCLIC( $n, k, H$ ) game.** Given an array  $A[0 \dots n-1]$  whose elements are an unknown cyclic permutation of  $\{0, \dots, n-1\}$ , the objective is to use  $k$  queries, at most  $H \leq k/2$  of which can be erroneous, so as to output an index of the array whose element is as small as possible.

► **Theorem 3.** *There is a questioner's strategy for MINCYCLIC( $n, k, H$ ) based on  $k$  comparison queries that outputs an index  $j$  such that  $A[j] \leq \lceil n \binom{k-H}{H} / 2^{k-H} \rceil$ , for all  $H \leq k/2$ .*

Last, we define a game that is related to searching in general permutations, and it will be useful in establishing lower bounds on the competitiveness of online bidding.

**SEARCH( $n, k, H$ ) game.** Given an array,  $A[0, \dots, n-1]$  whose elements are an unknown permutation of  $\{0, \dots, n-1\}$ , the objective is to use  $k$  queries, at most  $H$  of which can be erroneous, so as to output an index of the array whose element is as small as possible.

► **Theorem 4.** *For any questioner's strategy for the SEARCH( $n, k, H$ ) game, there is a responder's strategy such that if  $e$  is the element of  $A$  that is returned, then  $A[e] \geq \lfloor n \binom{k}{H} / 2^k \rfloor$ .*

### 3 A warm-up: Online time-series search

The *online (time series) search* problem formulates a simple, yet fundamental setting in decision-making under uncertainty. In this problem, a player must sell an indivisible asset within a certain time horizon, e.g., within a certain number of days  $d$ , that is unknown to the player. On each day  $i$ , a price  $p_i$  is revealed, and the player has two choices: either accept the price, and gain a profit  $p_i$  (at which point the game ends), or reject the price (at which point the game continues to day  $i+1$ ). If the player has not accepted a price by day  $d$ , then it accepts by default the last price  $p_d$ . The competitive ratio of the player's algorithm is the worst-case ratio, over all price sequences, of the maximum price in the sequence divided by the price accepted by the player.

The problem was introduced and studied in [19] that gave a simple, deterministic algorithm that achieves a competitive ratio equal to  $\sqrt{M/m}$ , where  $M, m$  are upper and lower bounds on the maximum and minimum price in the sequence, respectively, and which are assumed to be known to the algorithm. This bound is optimal for deterministic algorithms. Time-series search is a basic paradigm in online financial optimization, and several variants and

generalizations have been studied [17, 30, 40, 16]; see also the survey [27]. The problem has also been used as a case study for evaluating several performance measures of online algorithms, including measures alternative to competitive analysis [11, 1].

Time-series search was recently studied under the imperfect advice framework in [7], who showed an upper bound of  $(M/m)^{2^{2H-k/2}}$  on the competitive ratio with  $k$ -bit advice and tolerance  $H$ , under the assumption that  $H \leq k/4$ . Note that no upper bound is known for  $H \in (k/4, k/2]$ . If the advice is error-free, i.e., in the advice-complexity model, then a tight bound on the competitive ratio equal to  $(M/m)^{\frac{1}{2^{k+1}}}$  is due to [16].

We show the following result, as an application of the  $\text{FIND}(k, H)$  game discussed in Section 2.

► **Theorem 5.** *Consider the online time series search problem, with imperfect advice of size  $k$  and tolerance  $H \leq k/2$ . There is an algorithm that uses  $k$  comparison queries, and that has competitive ratio at most  $(M/m)^{\frac{1}{U+1}}$ , where  $U = \lfloor 2^{k-H} / \binom{k-H}{H} \rfloor$ , for any  $H \leq k/2$ . In contrast, no (deterministic) algorithm based on  $k$  subset queries has competitive ratio less than  $(M/m)^{\frac{1}{L+1}}$ , where  $L = \lceil 2^k / \binom{k}{H} \rceil$ .*

**Proof.** We first show the upper bound. Let  $a_1, \dots, a_U, r$  be defined such that  $r = \frac{a_1}{m} = \frac{a_2}{a_1} = \dots = \frac{a_U}{a_{U-1}} = \frac{M}{a_U}$ , hence  $r = (M/m)^{1/(U+1)}$ . The algorithm uses  $k$  comparison queries so as to find the best *reservation* price, in the set  $\{a_i\}_{i=1}^U$ , i.e., a threshold  $p$  above which the algorithm will always accept a price in the sequence. In particular, it can choose  $p$  to be the maximum value in  $\{a_i\}_{i=1}^U$  that does not exceed the maximum price in the sequence. This follows from Theorem 2, since  $U \leq 2^{k-H} / \binom{k-H}{H}$ . From the definition of the set  $\{a_i\}_{i=1}^U$ , it easily follows that this algorithm has competitive ratio at most  $r$ , which completes the proof of the upper bound.

We now show the lower bound. By way of contradiction, suppose that there is an algorithm  $A$  for time-series search with  $k$ -bit imperfect advice, and of competitive ratio less than  $C = (M/m)^{\frac{1}{L+1}}$ . We will show that  $A$  could then be used in the  $\text{FIND}(k, H)$  game so as to identify, using  $k$  queries, an unknown value in  $\{1, \dots, L+1\}$ , which is a contradiction to the upper bound of Theorem 2.

To arrive at the contradiction, define  $a_1, \dots, a_L$  and  $r'$  such that

$$r' = \frac{a_1}{m} = \frac{a_2}{a_1} = \dots = \frac{a_L}{a_{L-1}} = \frac{M}{a_L},$$

hence  $r' = (M/m)^{\frac{1}{L+1}} = C$ . Consider a game between the online algorithm  $A$  and the adversary, in which the request sequences consist of prices in  $\{m, a_1, \dots, a_L, M\}$ . More precisely, consider the set of request sequences of the form  $\sigma_i = m, a_1, \dots, a_i$ , for all  $i \in [1, L+1]$ , where  $a_{L+1}$  is defined to be equal to  $M$ . In  $\sigma_i$ ,  $A$  must accept price  $a_i$  to be strictly less than  $C$ -competitive. Equivalently,  $A$  uses  $k$  queries with at most  $H$  errors, and finds  $a_i$  in the set  $\{a_j\}_{j=1}^{L+1}$ , which contradicts Theorem 2. ◀

### 3.1 Comparison of the bounds

In order to compare the upper and lower bounds of Theorem 5, we need to be able to evaluate the partial sum of binomial coefficients. Since this partial sum does not have a closed form, we will rely on the following useful approximation from [32]. Let  $\mathcal{H}$  denote the *binary entropy* function. Then

$$\frac{2^{N\mathcal{H}(\frac{m}{N})}}{\sqrt{8m(1-\frac{m}{N})}} \leq \binom{N}{m} \leq 2^{N\mathcal{H}(\frac{m}{N})}, \quad \text{for } 0 < m < N/2. \quad (1)$$

We will also use the following property of the binary entropy function

$$4p(1-p) \leq \mathcal{H}(p) \leq (4p(1-p))^{1/\ln 4}, \quad \text{for all } p \in (0, 1). \quad (2)$$

We first show that the algorithm of Theorem 5 improves upon the one of [7]. First, note that [7] assumes that  $H \leq k/4$ , whereas Theorem 5 applies to all  $H \leq k/2$ . Furthermore, we improve on the competitive ratio for all values of  $H$  and  $k$ . For this, it suffices to show that  $\left(\binom{k-H}{H}\right)/2^{k-H} < 2^{2H-k/2}$ , which, from (1) holds if  $2^{(k-H)(\mathcal{H}(\frac{H}{k-H})-1)} < 2^{2H-k/2}$ , or equivalently  $(k-H)(\mathcal{H}(\frac{H}{k-H})-1) < 2H-k/2$ . Let  $\tau$  be such that  $\tau = H/k$  (hence  $\tau \leq 1/2$ ), then the latter is equivalent to showing that  $\mathcal{H}(\frac{\tau}{1-\tau}) < \frac{1+2\tau}{2-2\tau}$ . Using (2), it suffices to show that

$$\left(\frac{4\tau(1-2\tau)}{(1-\tau)^2}\right)^{1/\ln 4} < \frac{1+2\tau}{2-2\tau},$$

which holds for all  $\tau \leq 1/2$ .

Next, we investigate how close the upper and lower bounds of Theorem 5 are to each other. Recall that the bounds are of the form  $(M/m)^{1/(U+1)}$ , and  $(M/m)^{1/(L+1)}$ . Using (1), and ignoring for simplicity the floors and ceilings, we obtain that

$$U \geq 2^{k(1-\tau)(1-\mathcal{H}(\frac{\tau}{1-\tau}))} \quad \text{and} \quad L \leq \sqrt{8k\tau(1-\tau)}2^{k(1-\mathcal{H}(\tau))}.$$

The above inequalities, along with (2) show that the upper and lower bounds are very close to each other, since for any fixed value of  $\tau$ , we have that  $U \geq 2^{\Theta(k)}$  and  $L \leq 2^{\Theta(k)}$ .

## 4 Online bidding

Online bidding was introduced in [15] as a canonical problem for formalizing doubling-based strategies in online and offline optimization problems, such as searching for a target on the line, minimum latency, and hierarchical clustering. In this problem, a player wants to guess a hidden, unknown real value  $u \geq 1$ . To this end, the player defines an (infinite) sequence  $X = (x_i)$  of positive, increasing *bids*, which is called its *strategy*. The *cost of discovering* the hidden value  $u$  using the strategy  $X$ , denoted by  $c(X, u)$ , is defined to be equal to  $\sum_{i=1}^{j_u} x_i$ , where  $j_u$  is such that  $x_{j_u-1} < u \leq x_{j_u}$ . Hence one naturally defines the competitive ratio of the bidder's strategy  $X$  as  $\text{Cr}(X) = \sup_u \frac{c(X, u)}{u}$ .

In the standard version of the problem, i.e, assuming no advice, the doubling strategy  $x_i = 2^i$  achieves optimal competitive ratio equal to 4. Online bidding was studied under the untrusted advice model in [5], which gave bounds on the consistency/robustness tradeoffs. It was also studied under a model in which the prediction is the hidden value in [3, 5]. The problem is related to contract scheduling, studied in [6], see also the discussion in Section 4.1.3.

### 4.1 Online bidding with imperfect advice

#### 4.1.1 Upper bound

The idea behind the upper bound is as follows. We will consider bidding sequences from a space of  $2^k$  geometrically-increasing sequences (see Definition 6). In the ideal situation of perfect advice, the  $k$  advice bits could be used to identify the best strategy in this space. In the presence of advice errors, we will show how to exploit the cyclic structure of this space, in conjunction with our upper bound for the MINCYCLIC game (Theorem 3), so as to find a strategy that is not too far from the optimal.

We first define the space of geometrically-increasing bidding sequences.

► **Definition 6.** For given  $b > 1$ , and  $l \in \mathbb{N}^+$  define  $\mathcal{X}_{b,l}$  as the set of bidding sequences  $\{X_0, \dots, X_{l-1}\}$ , in which  $X_i = (b^{i+jl})_{j=0}^{\infty}$ , for all  $i \in [0, l-1]$ .

From the definition of  $\mathcal{X}_{b,l}$ , it is easy to see that for any potential target  $u$ , there is a cyclic permutation  $\pi$  of  $\{0, \dots, l-1\}$  which determines an ordering of the strategies in  $\mathcal{X}_{b,l}$  in terms of their performance. More precisely, suppose that  $X_{\pi(0)}$  is the best sequence that discovers  $u$  at least cost, say  $C$ . Then  $X_{\pi(i)}$  discovers  $u$  at cost at most  $b^i C$ . This property can help us show the following upper bound:

► **Theorem 7.** There is a bidding strategy based on  $k$  comparison queries of competitive ratio at most  $\frac{1+U}{2^k} \left(1 + \frac{2^k}{1+U}\right)^{1 + \frac{1+U}{2^k}}$ , where  $U = \lceil 2^H \binom{k-H}{H} \rceil$ .

### 4.1.2 Lower bound

The idea behind the lower bound is as follows. With  $k$  advice bits, the best one can do is choose the best strategy from a set  $\mathcal{X}$  that consists of at most  $2^k$  strategies. Note that if the advice were error-free,  $|\mathcal{X}|$  could be as large as  $2^k$ ; however, in the presence of errors, the algorithm may choose to narrow  $|\mathcal{X}|$ .

Our approach combines two ideas. The first idea uses the abstraction of the SEARCH( $n, k, H$ ) game, and the lower bound of Theorem 4. This result will allow us to place a lower bound on the rank of the chosen strategy, where the best strategy has rank 0. The second idea is to define a *measure* that relates how much worse a strategy of rank  $j$  in  $\mathcal{X}$  has to be relative to the best strategy in  $\mathcal{X}$ . We will accomplish this by appealing to the concepts of *parallelism* and *fault tolerance*.

More precisely, given integers  $p$ , and  $\phi$ , with  $\phi < p$ , we define the *fault-tolerant parallel bidding* problem, denoted by FPB( $p, \phi$ ), as follows. The player is allowed to run, in parallel,  $p$  bidding strategies; however,  $\phi$  of these strategies can be *faulty*, in that they never discover the target; e.g., we can think of a fault strategy as one in which the player abruptly stops submitting bids, at some point in time, akin to a “byzantine” failure. The cost of discovering a target  $u$  is then defined as the minimum cost at which one of the  $p - \phi$  non-faulty strategies discovers the target, noting that the faults are dictated by an adversary that aims to maximize this cost. The competitive ratio is defined accordingly.

The next theorem is the main technical result for FPB( $p, \phi$ ), which gives a lower bound on the competitive ratio of any strategy for this problem, as a function of the parameters  $p$ ,  $\phi$  and  $\alpha_{\bar{X}}$ . Here,  $\bar{X}$  is defined as the sorted sequence of all bids in the  $p$ -parallel strategy  $X$ , in non-decreasing order. Moreover, given a sequence  $X$  of positive reals, we define  $\alpha_X$  to be equal to  $\limsup_{i \rightarrow \infty} x_i^{1/i}$ .

► **Theorem 8.** Every  $p$ -parallel strategy  $X$  for FPB( $p, \phi$ ) has competitive ratio  $Cr(X) \geq \frac{\alpha_{\bar{X}}^{p+1+\phi}}{\alpha_X^{\phi-1}}$ .

**Proof sketch.** We use properties of  $p$ -parallel strategies so as to show that any such strategy satisfies  $Cr(X) \geq \sup_q \frac{\sum_{i=0}^{q+\phi-1} \bar{x}_i}{\sum_{i=q}^{q+(p-1)} \bar{x}_i}$ . We then use Gal’s functional theorem [21] to obtain the result. We omit several technical details. ◀

We now show how to obtain a lower bound for the problem by combining the above ideas. We emphasize a subtle point: unlike error-free advice of size  $k$ , where one should always choose the best strategy out of a collection of exactly  $2^k$  strategies, it is conceivable that, in the presence of errors, this collection could very well be of size  $l < 2^k$ . This is because, as  $l$  decreases, so does the effect of errors on the competitive ratio. In other words, we need to establish the result for *all* values  $l \leq 2^k$ , and not only for  $l = 2^k$ .

► **Theorem 9.** For every bidding sequence  $X$  and  $k$  subset queries in the imperfect advice model, we have  $\text{Cr}(X) \geq \frac{1}{L}(1+L)^{1+1/L}$ , where  $L = 2^k / \binom{k}{H}$ .

**Proof.** Every bidding strategy will use the query responses so as to select a strategy from a set  $\mathcal{X} = \{X_0, \dots, X_{l-1}\}$  of candidate sequences, for some  $l \leq 2^k$ . For a given target value  $u$ , there is an ordering of the  $l$  sequences in  $\mathcal{X}$  such that  $X_{\pi(i)}$  has no worse competitive ratio than  $X_{\pi(i+1)}$ , namely the permutation orders the sequences in decreasing order of performance. From Theorem 4, it follows that the strategy will choose a sequence  $X_j$  such that  $\pi(j) \geq \lfloor l \binom{k}{H} / 2^k \rfloor$ . The competitive ratio of the selected sequence is at least the competitive ratio of the  $l$ -parallel strategy defined by  $\mathcal{X}$ , in which up to  $\phi_l = \lfloor l \binom{k}{H} / 2^k \rfloor$  sequences may be faulty. From Theorem 8,

$$\text{Cr}(X) \geq \frac{\alpha_{\bar{X}}^{l+1+\phi_l}}{\alpha_{\bar{X}}^l - 1}, \quad \text{with } \phi_l = \lfloor l \binom{k}{H} / 2^k \rfloor. \quad (3)$$

We now consider two cases. Suppose first that  $l < L$ . In this case, case  $\phi_l = 0$ , and therefore (3) implies that  $\text{Cr}(X) \geq \alpha_{\bar{X}}^{l+1} / (\alpha_{\bar{X}}^l - 1)$ , which is minimized for  $\alpha_{\bar{X}} = (l+1)^{1/l} > 1$ , therefore  $\text{Cr}(X) \geq \frac{1}{l}(l+1)^{1+1/l}$ . This function is decreasing in  $l$ , and since  $l < L$  we have  $\text{Cr}(X) \geq \frac{1}{L}(1+L)^{1+1/L}$ . Next, suppose that  $l \in [L, 2^k]$ . In this case, (3) gives  $\text{Cr}(X) \geq \frac{\alpha_{\bar{X}}^{l(1+1/L)}}{\alpha_{\bar{X}}^l - 1}$ . The above expression is minimized for  $\alpha_{\bar{X}} = (1+L)^{1/l}$ , and by substitution we obtain again  $\text{Cr}(X) \geq \frac{1}{L}(1+L)^{1+1/L}$ . ◀

### 4.1.3 Comparison of the bounds

We can prove that the ratio between the two bounds is approximately

$$\log \frac{\text{UB}}{\text{LB}} \leq \frac{\sqrt{8k\tau(1-\tau)}k(1-\tau)(1-\mathcal{H}(\frac{\tau}{1-\tau}))}{2^{k(1-\tau)(1-\mathcal{H}(\frac{\tau}{1-\tau}))}} - \frac{k(1-\mathcal{H}(\tau))}{2^{k(1-\mathcal{H}(\tau))}},$$

where  $\tau = H/k$ . We infer that as  $k$  increases, and for any fixed value of  $\tau$ , the upper and lower bounds become very close to each other.

Note that the techniques of [6] imply an online bidding strategy with imperfect advice of competitive ratio roughly equal to  $f(2k/H)$ , where  $f$  is the decreasing function  $f(x) = \frac{1}{x}(1+x)^{1+\frac{1}{x}}$ . Thus, if  $H = \Theta(k)$ , then the competitive ratio is independent of the number of queries  $k$ . In contrast, the competitive ratio of Theorem 7 is roughly equal to  $f(2^k/U)$ , which is smaller than  $f(2k/H)$ , and which rapidly decreases as the number of queries  $k$  increases.

We also note that our analysis implies a tight bound on the advice complexity of online bidding. No previous bounds on the advice complexity of this problem were known.

## 5 Online fractional knapsack

In the online fractional knapsack problem, the request sequence consists of *items*, where item  $i$  has a *value*  $v_i \in \mathbb{R}^+$  and a *size*  $s_i \in (0, 1]$ . The algorithm has a knapsack of unit capacity, and when considering item  $i$ , it can accept irrevocably a fraction  $f_i \in (0, 1]$  of the item, subject to capacity constraints. More precisely, the algorithm aims to maximize  $\sum_i (f_i \cdot v_i)$  subject to  $\sum_i (f_i \cdot s_i) \leq 1$ . Online fractional knapsack has important applications in sponsored search auctions, ad allocation and online trading, and has been studied in several settings, e.g., [2, 24, 38, 14]. In this section, we study this problem in the imperfect advice setting.

Let  $d_i = v_i/s_i$  denote the *density* of item  $i$ . While the offline version of the problem admits an optimal solution via a simple greedy algorithm (that sorts all items by non-decreasing order of density, and accepts items in this order until the knapsack is full), the online version is more challenging. Suppose that  $d_i \in [L, U]$ , for  $L, U$  known to the algorithm. [13, 12] gave matching  $O(\log(U/L))$  and  $\Omega(\log(U/L))$  upper and lower bounds on the competitive ratio of the problem, respectively, and [41] showed an optimal bound of  $\ln(U/L) + 1$  for deterministic algorithms.

## 5.1 Upper bound

As in all previous work, we assume that the density of all items is in  $[L, U]$  for known values of  $L$  and  $U$ . Let  $d^*$  denote the smallest density of an item included at a positive fraction in the optimal solution. That is, the optimal algorithm OPT accepts a fraction 1 of items with density larger than  $d^*$ , and fills the remaining space with a fraction of items of density  $d^*$ . Unfortunately, knowing  $d^*$  (even its exact value) is not sufficient for an online algorithm to be anywhere as efficient as OPT. For example, an algorithm that accepts a fraction 1 of items of density larger than  $d^*$  has unbounded competitive ratio in sequences that consist only of items of density  $d^*$ . Similarly, an algorithm that accepts a fraction 1 of items with density at least  $d^*$  has unbounded competitive ratio in sequences in which items of density  $d^*$  appear early in the sequence, and items of greater density later in the sequence. However, if we denote by  $c^* \in (0, 1)$  the fraction of the knapsack in the optimal solution that is either empty or occupied with items of density  $d^*$ , then knowing the exact value of both  $d^*$  and  $c^*$  suffices to achieve optimality. Our approach will then aim to use  $k$  comparison queries so as to approximate  $c^*$  and  $d^*$ , then use these approximations to choose fractional items.

### 5.1.1 Algorithm and analysis

We describe the online algorithm. We first define two types of partitions, related to the parameters  $d^*$  and  $c^*$ . In what concerns  $d^*$ , partition the interval  $[L, U]$  into  $s$  sub-intervals  $I_1, \dots, I_s$  such that  $I_i = [d_{i-1}, d_i]$ , for  $s$  that will be specified later. We also set  $L = d_0, U = d_s$ . The values  $d_i$  are defined so that:  $\beta = \frac{d_1}{d_0} = \frac{d_2}{d_1} = \dots = \frac{d_s}{d_{s-1}}$ . Thus, we have  $\beta = (U/L)^{1/s}$  and  $d_i = L \cdot \beta^i$ , and note that  $d^* \in I_x$  for some  $x \in [1, s]$ .

In what concerns the parameter  $c^*$ , we partition the interval  $[0, 1]$  into  $m$  sub-intervals  $I'_1, \dots, I'_m$  such that  $I'_i = [c_{i-1}, c_i]$ ; we have  $c_0 = 0$  and  $c_m = 1$ . The value of  $m$  will be determined later; the values  $c_i$  are defined so that  $c_1 = c_2 - \frac{c_1}{\beta} = c_3 - \frac{c_2}{\beta} = \dots = c_m - \frac{c_{m-1}}{\beta}$ .

It readily follows that for  $i \geq 1$ , we have  $c_i = \frac{\beta^{m+i-1} - \beta^{m+i-2}}{\beta^m - 1}$ . In particular,  $c_1 = \frac{\beta^m - \beta^{m-1}}{\beta^m - 1}$ , and  $\frac{1}{1-c_1} = \frac{\beta^m - 1}{\beta^{m-1} - 1}$ . Note also that  $c^* \in I'_y$  for some  $y \in [1, m]$ .

Provided that  $s \cdot m \leq \lfloor 2^{k-H} / \binom{k-H}{H} \rfloor$ , Theorem 2 shows that the algorithm can use  $k$  comparison queries so as to identify both  $x$  and  $y$ . Given these values, the algorithm reserves, in its knapsack, a capacity  $c = c_{y-1}$  for items with density in the range  $I_x = [d_{x-1}, d_x]$ , to which we refer as *critical items*. The algorithm uses the remaining capacity of  $1 - c$  for items of density larger than  $d_x$ , to which we refer as *heavy items*, and accepts a fraction 1 of all critical items, as long as the capacity  $c$  reserved for them allows. Similarly, the algorithm accepts a fraction 1 of heavy items and places them in their dedicated space of the knapsack. Given that  $c^* \in I'_y$ , we have  $1 - c > 1 - c^*$ ; that is, the reserved capacity for heavy items is at least equal to the total size of these items. In other words, the algorithm can afford to accept all heavy items. The algorithm rejects all items of density smaller than  $d_{x-1}$ .

► **Theorem 10.** *For any  $H \leq k/2$ , the above algorithm has competitive ratio*

$$\min_{s,m \in \mathbb{N}} f_m(\beta) \quad \text{where} \quad \beta = (U/L)^{1/s}, \quad \text{and} \quad f_m(\beta) = \frac{\beta^m - 1}{\beta^{m-1} - 1}$$

$$\text{subject to} \quad s \cdot m \leq \lfloor 2^{k-H} / \binom{k-H}{H} \rfloor.$$

## 5.2 Lower bound

We will show a lower bound  $C(k, H)$  on the competitive ratio of any algorithm with imperfect advice. For the sake of contradiction, suppose there is an algorithm  $A$  of competitive ratio better than  $C(k, H)$ . Our proof is based on a reduction from the  $\text{FIND}(k, H)$  game. Specifically, we prove that, based on  $A$ , we obtain a questioner's strategy for  $\text{FIND}(k, H)$  which can find a value  $z \in \{1, \dots, p\}$ , with  $p = \lceil 2^k / \binom{k}{H} \rceil + 1$ , which contradicts Theorem 2.

We give the intuition behind the proof. Let  $s$  and  $m$  be any two positive integers such that  $s \cdot m \leq p$  and  $s \cdot (m+1) > p$ . Define  $\beta = (U/L)^{1/s}$ , and  $d_i = U \cdot \beta^i$ , for  $i \in [1, s]$ . Given a pair  $(x, y)$  of integers, where  $x \in \{1, \dots, s\}$  and  $y \in \{1, \dots, m+1\}$ , define the sequence

$$\sigma_{x,y} = ((d_1, 1), (d_2, 1), \dots, (d_{x-1}, 1), (d_x, c_y)),$$

where  $(d_i, j)$  indicates a subsequence of  $j/\epsilon$  items, each of which has size  $\epsilon$  and density  $d_i$ , and where  $\epsilon$  is infinitesimally small.  $c_y \in [0, 1]$  is defined appropriately in the proof. For this sequence,  $\text{OPT}(\sigma_{x,y}) = (1 - c_y)d_{x-1} + c_y d_x$ . There are  $s \cdot (m+1) > p$  such sequences, and  $\sigma_{x,y}$  is a prefix sequence of  $\sigma_{x,y+1}$ , and  $\sigma_{x,m}$  is a prefix sequence of  $\sigma_{x+1,1}$ . In the proof, we consider request sequences of this form, and we show that if  $A$  is  $C(k, H)$ -competitive, its decisions can help find any given  $z \in \{1, \dots, p\}$ , which contradicts Theorem 2.

► **Theorem 11.** *For the fractional knapsack problem, where items densities are in  $[L, U]$ , no deterministic algorithm with  $k$  subset queries, out of which  $H \leq k/2$  may have erroneous responses, can achieve a competitive ratio better than*

$$C(k, H) = \min_{s,m \in \mathbb{N}} g_m(\beta) \quad \text{where} \quad \beta = (U/L)^{1/s}, \quad g_m(\beta) = \left( \frac{\beta^2 - \beta + 1}{2\beta + 1} \right)^{1/(m+1)}$$

$$\text{subject to} \quad s \cdot m \leq \lceil 2^k / \binom{k}{H} \rceil + 1.$$

### Comparison of the bounds

Let  $\tau = H/k$ . Since  $\frac{\beta^m - 1}{\beta^{m-1} - 1} \leq \beta$ , using (1), the upper bound of Theorem 10 is at most  $(U/L)^q$ , where  $q \leq 1/2^{k(1-\tau)(1-\mathcal{H}(\frac{\tau}{1-\tau}))}$ . Furthermore, since  $\frac{\beta^2 - \beta + 1}{2\beta + 1} \geq \frac{\beta}{3}$  (for all  $\beta \geq 3$ ), the lower bound of Theorem 11 is at least  $(U/L)^{q'} (1/3)^{q'}$ , where  $q' \geq 1/(2\sqrt{8k\tau(1-\tau)}2^{k(1-\mathcal{H}(\tau))} + 1)$ , for all  $U/L \geq 3$ . For simplicity, we omitted the floors and ceilings.

## 6 Waiving the assumption of the tolerance parameter

In the imperfect advice setting we have studied so far, the algorithm defines an application-specific tolerance parameter that measures its desired tolerance to errors (or equivalently, an anticipated upper bound on the error). This parameter is in a sense required, since the analysis of Rényi-Ulam games in [36] involves the extreme value of error (i.e.,  $H$ ) instead of the instance-specific error value (i.e.,  $\eta$ ). Nevertheless, in this section, we discuss how to mitigate the need for pre-determining a tolerance parameter. We propose two different

approaches, based on *resource-augmentation*, and *robustification*, which we discuss in what follows. We use the time-series search and online bidding problems as illustrations, even though our approach may carry through in other online problems, at the expense of more complex calculations.

### 6.1 Resource augmentation

In this setting, we compare an *oblivious* online algorithm  $A$  with  $l$  advice bits and no information on the error bound, to an online algorithm  $B$  that has  $k$  *ideal* (i.e. error-free) advice bits. Specifically, we are interested in finding the smallest  $l \geq k$  (as a function of  $k$ ) for which algorithm  $A$  is at least as good as algorithm  $B$ , regardless of the advice error of  $A$ .

The following theorem shows that  $O(1)$ -factor resource augmentation suffices to obtain an oblivious algorithm that is at least as efficient as any algorithm that operates in the ideal setting of error-free advice, and even if a fraction  $1/3 - c$  of the advice bits may be erroneous, for any constant  $c$ .

► **Theorem 12.** *Consider the time-series and the online bidding problems. For all sufficiently large  $k$ , and any  $c \in (0, 1/3)$ , there is an oblivious online algorithm  $A$  with advice of size  $l$ , whose competitive ratio is at least as good as that of any online algorithm  $B$  with  $k$  bits of perfect (i.e. error-free) advice, where  $l = \frac{1}{(\frac{2}{3}+c)(1-\mathcal{H}(\frac{\frac{1}{3}-c}{\frac{2}{3}+c}))}k + 1$ , for any error  $\eta \leq (1/3 - c)l$  in the advice of  $A$ .*

### 6.2 Robustification

In this setting, we augment the imperfect advice framework by requiring not only that the algorithm minimizes the competitive ratio assuming that the advice error is at most the tolerance  $H$ , but also that its competitive ratio does not exceed a *robustness* requirement  $r$ , for some specified  $r$ , if the error exceeds  $H$  (and in particular, if the advice is adversarially generated). We call such online algorithms *r-robust*. Thus, this model can be seen as an extension of both the imperfect advice and the untrusted advice model of [5].

For the time-series problem, we obtain the following result, which generalizes Theorem 5. In particular, note that Theorem 5 is a special case of Theorem 13 for  $\rho = 1$ .

► **Theorem 13.** *Consider the online time series search problem, with imperfect advice of size  $k$ , tolerance  $H \leq k/2$ , and robustness  $r = (M/m)^\rho$ , where  $\rho \in (1/2, 1]$ . There is an  $r$ -robust algorithm that uses  $k$  comparison queries, and has competitive ratio at most  $(M/m)^{\frac{2\rho-1}{\rho+1}}$ , where  $U = \lfloor 2^{k-H} / \binom{k-H}{H} \rfloor$ , for any  $H \leq k/2$ . Moreover, no (deterministic) algorithm based on  $k$  subset queries has competitive ratio better than  $(M/m)^{\frac{2\rho-1}{L+1}}$ , where  $L = \lceil 2^k / \binom{k-H}{H} \rceil$ .*

The analysis of  $r$ -robust algorithms for online bidding is more challenging, in particular in what concerns the impossibility results. We give an overview of the approach. For the upper bound, we can follow an analysis along the lines of Theorem 7, however, each bidding sequence in the collection  $\mathcal{X}_{b,2^k}$  must be individually  $r$ -robust. This is easy to enforce, and it requires that  $b$  must be such that  $b^2/(b-1) \leq r$ . The lower bound is more subtle: the proof follows the lines of Theorem 9, but uses the fact that if all the  $l$  sequences in  $X_0, \dots, X_{l-1}$  must be  $r$ -robust, then  $\alpha_{\bar{X}}^2/(\alpha_{\bar{X}} - 1) \leq r$ . We obtain the following:

► **Theorem 14.** *For every  $r \geq 4$  there is an  $r$ -robust bidding strategy with  $k$ -bit imperfect advice that has competitive ratio at most*

$$\min_{b>1} \frac{b^{2^k+U+1}}{b^{2^k}-1}, \quad \text{subject to } b^{2^{k+1}}/(b^{2^k}-1) \leq r, \quad \text{and where } U = \lceil 2^H \binom{k-H}{H} \rceil.$$

Furthermore, every  $r$ -robust bidding strategy has competitive ratio at least

$$\min_{\alpha>1} \frac{\alpha^{2^k+L+1}}{\alpha^{2^k}-1} \quad \text{subject to } \alpha^{2^k}/(\alpha^k-1) \leq r, \quad \text{and where } L = \lfloor \binom{k}{H} \rfloor.$$

---

## References

- 1 Iftikhar Ahmad, Marcus Pirron, and Günter Schmidt. Analysis of threat based algorithm using different performance measures. *RAIRO: Operations Research*, 55:2393, 2021.
- 2 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and gap in the random order model. *Algorithmica*, 83(6):1750–1785, 2021.
- 3 Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. A regression approach to learning-augmented online algorithms. *Advances in Neural Information Processing Systems*, 34:30504–30517, 2021.
- 4 Spyros Angelopoulos. Online search with a hint. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 51:1–51:16, 2021.
- 5 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *Proceedings of the 11th International Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 52:1–52:15, 2020.
- 6 Spyros Angelopoulos and Shahin Kamali. Contract scheduling with predictions. *Journal of Artificial Intelligence Research*, 77:395–426, 2023.
- 7 Spyros Angelopoulos, Shahin Kamali, and Dehou Zhang. Online search with best-price and query-based predictions. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, pages 9652–9660, 2022.
- 8 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, and Richard Královic. On the advice complexity of the  $k$ -server problem. *J. Comput. Syst. Sci.*, 86:159–170, 2017.
- 9 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. Online algorithms with advice: The tape model. *Inf. Comput.*, 254:59–83, 2017.
- 10 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.
- 11 Joan Boyar, Kim S. Larsen, and Abyayananda Maiti. A comparison of performance measures via online search. *Theoretical Computer Science*, 532:2–13, 2014.
- 12 Niv Buchbinder and Joseph Naor. Improved bounds for online routing and packing via a primal-dual approach. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 293–304. IEEE, 2006.
- 13 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.
- 14 Ying Cao, Bo Sun, and Danny Tsang. Optimal online algorithms for one-way trading and online knapsack problems: A unified competitive analysis. In *Proceedings of the 59th IEEE Conference on Decision and Control (CDC)*, pages 1064–1069, 2020.
- 15 Marek Chrobak and Claire Kenyon-Mathieu. SIGACT news online algorithms column 10: Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.
- 16 Jhoirene Clemente, Juraj Hromkovič, Dennis Komm, and Christian Kudahl. Advice complexity of the online search problem. In *Proceedings of the 27th International Workshop on Combinatorial Algorithms (IWOCA)*, pages 203–212, 2016.

- 17 Peter Damaschke, Phuong Hoai Ha, and Philippas Tsigas. Online search with time-varying price bounds. *Algorithmica*, 55(4):619–642, 2009.
- 18 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Theor. Informatics Appl.*, 43(3):585–613, 2009.
- 19 Ran El-Yaniv, Amos Fiat, Richard M Karp, and Gordon Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 2001.
- 20 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.
- 21 Shmuel Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.
- 22 Sungjin Im, Ravi Kumar, Aditya Petety, and Manish Purohit. Parsimonious learning-augmented caching. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pages 9588–9601. PMLR, 2022.
- 23 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Online knapsack with frequency predictions. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2733–2743, 2021.
- 24 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing LPs in the random-order model. *SIAM J. Comput.*, 47(5):1939–1964, 2018.
- 25 Dennis Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.
- 26 Russell Lee, Jessica Maghachian, Mohammad H. Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. Online peak-aware energy scheduling with untrusted advice. In *Proceedings of the 12th ACM International Conference on Future Energy Systems (eEnergy)*, pages 107–123. ACM, 2021.
- 27 Bin Li and Steven CH Hoi. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 46(3):1–36, 2014.
- 28 Tongxin Li, Ruixiao Yang, Guannan Qu, Guanya Shi, Chenkai Yu, Adam Wierman, and Steven H. Low. Robustness and consistency in linear quadratic control with untrusted predictions. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(1):18:1–18:35, 2022.
- 29 Alexander Lindermayr and Nicole Megow. Repository of works on algorithms with predictions. <https://algorithms-with-predictions.github.io/about>, 2023. Accessed: 2023-04-01.
- 30 Julian Lorenz, Konstantinos Panagiotou, and Angelika Steger. Optimal algorithms for k-search with application in option pricing. *Algorithmica*, 55(2):311–328, 2009.
- 31 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021.
- 32 Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.
- 33 Arya Mazumdar and Barna Saha. Clustering with noisy queries. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS)*, pages 5788–5799. NIPS, 2017.
- 34 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020.
- 35 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pages 9661–9670, 2018.
- 36 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winklmann, and Joel Spencer. Coping with errors in binary search procedures. *J. Comput. Syst. Sci.*, 20(3):396–404, 1980.
- 37 Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
- 38 Bo Sun, Ali Zeynali, Tongxin Li, Mohammad Hajiesmaili, Adam Wierman, and Danny HK Tsang. Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(3):1–32, 2020.

- 39 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- 40 Yinfeng Xu, Wenming Zhang, and Feifeng Zheng. Optimal algorithms for the online time series search problem. *Theoretical Computer Science*, 412(3):192–197, 2011.
- 41 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *Proceedings of the International Workshop on Internet and Network Economics (WINE)*, pages 566–576. Springer, 2008.



# Multivariate to Bivariate Reduction for Noncommutative Polynomial Factorization

Vikraman Arvind  

The Institute of Mathematical Sciences (HBNI), Chennai, India  
Chennai Mathematical Institute, Siruseri, Kelambakkam, India

Pushkar S. Joglekar 

Vishwakarma Institute of Technology, Pune, India

---

## Abstract

---

Based on a theorem of Bergman [6, Theorem 4.5.3] we show that multivariate noncommutative polynomial factorization is deterministic polynomial-time reducible to the factorization of bivariate noncommutative polynomials. More precisely, we show the following:

1. In the white-box setting, given an  $n$ -variate noncommutative polynomial  $f \in \mathbb{F}\langle X \rangle$  over a field  $\mathbb{F}$  (either a finite field or the rationals) as an arithmetic circuit (or algebraic branching program), computing a complete factorization of  $f$  into irreducible factors is deterministic polynomial-time reducible to white-box factorization of a noncommutative bivariate polynomial  $g \in \mathbb{F}\langle x, y \rangle$ ; the reduction transforms  $f$  into a circuit for  $g$  (resp. ABP for  $g$ ), and given a complete factorization of  $g$  (namely, arithmetic circuits (resp. ABPs) for irreducible factors of  $g$ ) the reduction recovers a complete factorization of  $f$  in polynomial time.

We also obtain a similar deterministic polynomial-time reduction in the black-box setting.

2. Additionally, we show over the field of rationals that bivariate linear matrix factorization of  $4 \times 4$  matrices is at least as hard as factoring square-free integers. This indicates that reducing noncommutative polynomial factorization to linear matrix factorization (as done in [1]) is unlikely to succeed over the field of rationals even in the bivariate case. In contrast, multivariate linear matrix factorization for  $3 \times 3$  matrices over rationals is in polynomial time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Algebraic complexity theory

**Keywords and phrases** Arithmetic circuits, algebraic branching programs, polynomial factorization, automata, noncommutative polynomial ring

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.14

**Acknowledgements** We thank the reviewers for insightful suggestions and comments.

## 1 Introduction

The main aim of this paper is to show that multivariate polynomial factorization in the free noncommutative ring  $\mathbb{F}\langle x_1, x_2, \dots, x_n \rangle$  is polynomial-time reducible to *bivariate* noncommutative polynomial factorization in the bivariate ring  $\mathbb{F}\langle x, y \rangle$ . Such a result for commutative polynomial factorization is well-known due to Kaltofen's seminal work [9, 10] on multivariate polynomial factorization in the commutative polynomial ring  $\mathbb{F}[y_1, y_2, \dots, y_n]$ . However, this problem was open for noncommutative polynomials. Recently, a randomized polynomial-time algorithm was obtained for the factorization of noncommutative polynomials over finite fields, where the input polynomial is given by a noncommutative formula [1].<sup>1</sup> Broadly speaking, the algorithm of [1] works via Higman linearization ([8] [6] [7]) and reduces the problem to linear matrix factorization which turns out to have a randomized polynomial-time algorithm over finite fields.

---

<sup>1</sup> Factorization of *homogeneous* noncommutative polynomials is easier as it can be reduced to factorization of a special case of commutative polynomials. See [4] for details.



© Vikraman Arvind and Pushkar S. Joglekar;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 14; pp. 14:1–14:15

Leibniz International Proceedings in Informatics



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

► **Problem 1** (Linear Matrix Factorization Problem). *The linear matrix factorization problem over a field  $\mathbb{F}$  takes as input a linear matrix:  $L = A_0 + \sum_{i=1}^n A_i x_i$ , where the  $A_i$  are  $d \times d$  scalar matrices (over  $\mathbb{F}$ ), the  $x_i, 1 \leq i \leq n$  are noncommuting variables, and  $A_0$  is assumed invertible for technical reasons. The problem is to compute a factorization of  $L$  as a product of irreducible linear matrices.*

The study of matrix factorization (linear matrix factorization, in particular) is an important part of Cohn's factorization theory over general free ideal rings. [6, 5].

Coming back to the polynomial factorization algorithm described in [1], the algorithm reduces polynomial factorization to linear matrix factorization which is, in turn, reducible to the problem of computing a common invariant subspace for a collection of  $n$  matrices. The common invariant subspace problem over finite fields can be efficiently solved using Ronyai's algorithm [12] which is based on the Artin-Wedderburn theorem for decomposition of algebras. This approach, however, runs into serious difficulties over rationals. Given a simple matrix algebra<sup>2</sup>  $\mathcal{A}$  over rationals, we do not know an efficient algorithm for checking if  $\mathcal{A}$  is a division algebra or whether it has zero divisors. This is one of our motivations for obtaining a reduction from multivariate polynomial factorization to bivariate factorization. Because Higman Linearization of a bivariate noncommutative polynomial given by a formula will yield a bivariate linear matrix. One could hope that factorization of a bivariate linear matrix is computationally easier than factorization of an  $n$ -variate linear matrix. Unfortunately, this is not the case. As we will see, even for 4-dimensional bivariate linear matrices the problem of factorization is at least as hard as factoring square-free integers.

### Multivariate to Bivariate

We start with some formal preliminaries. Let  $\mathbb{F}$  be any field and  $X = \{x_1, x_2, \dots, x_n\}$  be a set of  $n$  free noncommuting variables. Let  $X^*$  denote the set of all free words (which are monomials) over the alphabet  $X$  with concatenation of words as the monoid operation and the empty word  $\epsilon$  as identity element.

The *free noncommutative ring*  $\mathbb{F}\langle X \rangle$  consists of all finite  $\mathbb{F}$ -linear combinations of monomials in  $X^*$ , where the ring addition  $+$  is coefficient-wise addition and the ring multiplication  $*$  is the usual convolution product. More precisely, let  $f, g \in \mathbb{F}\langle X \rangle$  and let  $f(m) \in \mathbb{F}$  denote the coefficient of monomial  $m$  in polynomial  $f$ . Then we can write  $f = \sum_m f(m)m$  and  $g = \sum_m g(m)m$ , and in the product polynomial  $fg$  for each monomial  $m$  we have  $fg(m) = \sum_{m_1 m_2 = m} f(m_1)g(m_2)$ . The *degree* of a monomial  $m \in X^*$  is the length of the monomial  $m$ , and the degree  $\deg f$  of a polynomial  $f \in \mathbb{F}\langle X \rangle$  is the degree of a largest degree monomial in  $f$  with nonzero coefficient. For polynomials  $f, g \in \mathbb{F}\langle X \rangle$  we clearly have  $\deg(fg) = \deg f + \deg g$ .

A *nontrivial factorization* of a polynomial  $f \in \mathbb{F}\langle X \rangle$  is an expression of  $f$  as a product  $f = gh$  of polynomials  $g, h \in \mathbb{F}\langle X \rangle$  such that  $\deg g > 0$  and  $\deg h > 0$ . A polynomial  $f \in \mathbb{F}\langle X \rangle$  is *irreducible* if it has no nontrivial factorization and is *reducible* otherwise. For instance, all degree 1 polynomials in  $\mathbb{F}\langle X \rangle$  are irreducible. Clearly, by repeated factorization every polynomial in  $\mathbb{F}\langle X \rangle$  can be expressed as a product of irreducibles.

The problem of *noncommutative polynomial identity testing* (PIT) for multivariate polynomials is known to easily reduce to noncommutative PIT for bivariate polynomials: the reduction is given by the  $x_i \rightarrow xy^i, 1 \leq i \leq n$ , which transforms a given arithmetic circuit (or formula or algebraic branching program) computing a polynomial  $f(x_1, x_2, \dots, x_n)$  to the

<sup>2</sup> i.e. the algebra has no nontrivial two-sided ideals.

bivariate polynomial  $g(x, y) = f(xy, xy^2, \dots, xy^n)$ . As this substitution map ensures that every monomial of  $f$  is mapped to a distinct monomial of  $g(x, y)$ ,  $f$  is the zero polynomial if and only if  $g(x, y)$  is the zero polynomial. Indeed, this map even gives an injective homomorphism from the ring  $\mathbb{F}\langle x_1, x_2, \dots, x_n \rangle$  to  $\mathbb{F}\langle x, y \rangle$  [6, Problem 14, Exercises 2.5]. However, it does not preserve factorizations. For example, the polynomial  $f = x_3x_1 + x_4x_2 + x_4x_1 + x_5x_2 \in \mathbb{F}\langle X \rangle$  is clearly irreducible. But the image of  $f$  under this map has the nontrivial factorization  $(xy^2 + xy^3)(yxy + y^2xy^2)$ . Thus, it cannot be used to obtain a reduction from noncommutative multivariate polynomial factorization to bivariate polynomial factorization.

### Bergman's 1-inert embedding

However, based on a theorem of Bergman [6, Theorem 4.5.3], we can obtain a polynomial-time reduction from factorization of multivariate noncommutative polynomials in  $\mathbb{F}\langle x_1, x_2, \dots, x_n \rangle$  given by arithmetic circuits (resp. noncommutative algebraic branching programs (ABP)) to factorization of bivariate noncommutative polynomials in  $\mathbb{F}\langle x, y \rangle$ , again given by arithmetic circuit (resp. an ABP). This reduction is polynomial-time bounded for both finite fields and rationals. In the case of rationals we need to ensure that the bit complexities of all numbers involved are polynomially bounded. Furthermore, we show that essentially the same reduction works in the black-box setting as well.

The notion of 1-inert embeddings is defined below for free noncommutative polynomials.

► **Definition 2 (1-inert embedding).** [5] *Let  $X_\infty = \{x_1, x_2, \dots\}$  be a countably infinite set of free noncommuting variables and  $\{x, y\}$  be two free noncommuting variables. A 1-inert embedding of  $\mathbb{F}\langle X_\infty \rangle$  into  $\mathbb{F}\langle x, y \rangle$  is an injective homomorphism  $\varphi : \mathbb{F}\langle X \rangle \rightarrow \mathbb{F}\langle x, y \rangle$  such that for each polynomial  $f \in \mathbb{F}\langle X \rangle$ , if its image  $\varphi(f)$  factorizes nontrivially in  $\mathbb{F}\langle x, y \rangle$  as  $\varphi(f) = g_1 \cdot g_2$  then their preimages  $\varphi^{-1}(g_1)$  and  $\varphi^{-1}(g_2)$  exist and, since  $\varphi$  is a homomorphism, it gives a nontrivial factorization  $f = \varphi^{-1}(g_1)\varphi^{-1}(g_2)$  of  $f$  in  $\mathbb{F}\langle X \rangle$ .*

► **Remark 3.** The above definition implies that for all factorizations  $\varphi(f) = g_1g_2$ , the polynomials  $g_1$  and  $g_2$  are in the range of  $\varphi$ . Cohn [6, 5] treats 1-inert embeddings  $\varphi : R_1 \rightarrow R_2$  for general noncommutative integral domains  $R_1$  and  $R_2$ , which we do not require for our results.

► **Definition 4.** *A complete factorization of noncommutative polynomial  $f \in \mathbb{F}\langle X \rangle$  is a factorization  $f = f_1 \cdot f_2 \cdots f_r$  into a product of irreducible polynomials  $f_i \in \mathbb{F}\langle X \rangle$ .*

Given an algebraic branching program (resp. Arithmetic Circuit) for  $f$ , we can efficiently obtain an algebraic branching program (resp. Arithmetic Circuit) for  $\varphi(f)$  and then we use idea of running a substitution automata on ABPs or circuits (see e.g. [4], [2], [3]) to construct a complete factorization of  $f$  given a complete factorization of  $\varphi(f)$ . In the next section we will elaborate and expand upon Bergman's embedding theorem [5] and show how to get an effective algorithmic version which is useful for our purpose of reconstruction of factors of  $f$  from factors of  $\varphi(f)$ .

The rest of the paper is organized as follows: In Section 2 we give necessary details of Bergman's result. In Section 3 we present the reductions. Motivated by the connection between noncommutative polynomial factorization and linear matrix factorization, in Section 4 we show a hardness result for bivariate linear matrix factorization for  $4 \times 4$  linear matrices over rationals. In contrast we obtain an efficient linear matrix factorization algorithm for  $3 \times 3$  linear matrices over rationals.

## 2

 Bergman's embedding

We recall the *graded lexicographic ordering*  $\prec$  on monomials in  $\{x, y\}^*$ , which is a total ordering on  $\{x, y\}^*$  defined as follows:

For monomials  $m_1, m_2 \in \{x, y\}^*$ ,  $m_1 \neq m_2$ , we say  $m_1 \prec m_2$  if either  $\deg(m_1) < \deg(m_2)$  or  $\deg(m_1) = \deg(m_2)$  and in the leftmost position  $i$  where they differ we have  $m_1[i] = y$  and  $m_2[i] = x$ .

For any polynomial  $g$ , let  $\text{supp}(g)$  denote the set of all monomials of  $g$  with non-zero coefficient. When  $m_1 \prec m_2$  we say that monomial  $m_1$  is *smaller* than monomial  $m_2$ . Equivalently, we say  $m_2$  is *larger* than  $m_1$ . The *leading monomial* of a polynomial  $g \in \mathbb{F}\langle x, y \rangle$  is the monomial  $m \in \text{supp}(g)$  (denoted by  $\text{lm}(g)$ ) such that  $w \prec m$  for all  $w \in \text{supp}(g)$ . That is, the leading monomial of  $g$  is the largest monomial in  $\text{supp}(g)$ .

For a monomial  $m \in \{x, y\}^*$  let  $d_x(m)$  (resp.  $d_y(m)$ ) denote the number of occurrences of  $x$  (resp.  $y$ ) in  $m$ . The *imbalance*  $i(m)$  of the monomial  $m$  is defined as

$$i(m) = d_x(m) - d_y(m).$$

Let  $B \subset \mathbb{F}\langle x, y \rangle$  be the set of all polynomials  $f$  such that every monomial  $m \in \text{supp}(f)$  has imbalance zero, i.e.  $i(m) = 0$  for all  $m \in \text{supp}(f)$ . Clearly,  $B$  is a subalgebra of  $\mathbb{F}\langle x, y \rangle$ . Let  $T$  be the set of all *minimally balanced* monomials. That is, for  $m \in T$  either  $m = \epsilon$  or  $i(m) = 0$  and for any proper prefix  $m'$  of  $m$  such that  $m' \neq \epsilon$ ,  $i(m') > 0$ . Notice that for all monomials  $m \in T \setminus \{\epsilon\}$  its leftmost symbol  $m[1]$  is  $x$ . We arrange the nontrivial monomials in  $T$  in increasing  $\prec$ -ordering. Let  $u_i$  denote the  $i^{\text{th}}$  monomial in this ordering. Let  $\bar{u}_i$  be the monomial obtained from  $u_i$  by replacing every occurrence of  $x$  by  $y$  and  $y$  by  $x$ . Let  $\bar{T} = \{\bar{u}_i \mid i \geq 1\}$ . It is easy to see that the monomials in  $T \cup \bar{T}$  generate the algebra  $B$ . In fact, every monomial  $m \in B$  is *uniquely* expressible as a product  $g_1 g_2 \dots g_\ell$ , where each  $g_j \in T \cup \bar{T}$ . If  $g_j \in T$  it is a  $T$ -factor of  $m$  and if  $g_j \in \bar{T}$  it is a  $\bar{T}$ -factor of  $m$ . Let  $C$  be the subalgebra of  $B$  generated by  $\{u_i + \bar{u}_i \mid i \geq 1\}$ .

► **Lemma 5.** *Let  $B$  and  $C$  be the subalgebras of  $\mathbb{F}\langle x, y \rangle$  as defined above.*

- *The leading monomial  $m$  of any polynomial in  $C$  has the form  $m = u_{i_1} u_{i_2} \dots u_{i_\ell}$ , where each  $u_{i_j}$  is a  $T$ -factor. That is,  $m$  does not have any  $\bar{T}$ -factor.*
- *Every polynomial  $f \in B \setminus C$  can be written as  $f = g + h$  for some  $g \in C$  and  $h \in B$ , such that the leading monomial of  $h$  has a  $\bar{T}$ -factor.*

**Proof.** By definition, each  $g \in C$  is a linear combination of products of the form  $\prod_{k=1}^\ell (u_{i_k} + \bar{u}_{i_k})$ . Hence, if  $\text{supp}(g)$  contains the monomial  $v_1 v_2 \dots v_\ell$ , where  $v_k \in \{u_{j_k}, \bar{u}_{j_k}\}$  for  $k \in [\ell]$ , then  $\text{supp}(g)$  also contains the degree- $d$  monomial  $u_{j_1} u_{j_2} \dots u_{j_\ell}$  (in fact, with the same coefficient as  $v_1 v_2 \dots v_\ell$ ). If  $u_{j_1} u_{j_2} \dots u_{j_\ell} \neq v_1 v_2 \dots v_\ell$  then, by definition of  $\prec$ , the monomial  $u_{j_1} u_{j_2} \dots u_{j_\ell}$  is larger than  $v_1 v_2 \dots v_\ell$ . Therefore, the leading monomial of any polynomial  $g \in C$  has the form claimed.

Next, let  $f \in B \setminus C$ . We will show the second part of the lemma by induction on the leading monomial of  $f$  w.r.t. the  $\prec$ -ordering (which is a well ordering on monomials).

The base case of the induction is when the leading monomial of  $f$  has a  $\bar{T}$ -factor then the claim follows as  $f = 0 + f$  and  $0 \in C$ . Suppose the leading monomial of  $f$  is  $m = u_{j_1} u_{j_2} \dots u_{j_\ell}$ . If the coefficient of  $m$  in  $f$  is  $\alpha \neq 0$ , let

$$f_1 = f - \alpha(u_{j_1} + \bar{u}_{j_1})(u_{j_2} + \bar{u}_{j_2}) \dots (u_{j_\ell} + \bar{u}_{j_\ell}). \quad (1)$$

If  $m_1$  is the leading monomial of  $f_1$  then clearly  $m_1 \prec m$ . Furthermore,  $f_1 \in B \setminus C$  as  $f - f_1 \in C$ . By induction hypothesis, we have  $f_1 = g' + h$  such that  $g' \in C$  and the leading monomial of  $h$  has a  $\bar{T}$ -factor. Since  $f = (f - f_1) + g' + h$  and  $g = (f - f_1) + g' \in C$ , this completes the induction and the proof. ◀

Let  $X_\infty = \{x_1, x_2, \dots\}$  be a countably infinite set of free noncommuting indeterminates. Consider the mapping  $\varphi : \mathbb{F}\langle X_\infty \rangle \mapsto \mathbb{F}\langle x, y \rangle$  defined as follows:

- Let  $\varphi(x_i) = u_i + \bar{u}_i$  for all  $x_i \in X_\infty$ .
- Extend  $\varphi$  to all monomials by multiplication. That is,  $\varphi(x_{i_1}x_{i_2}\dots x_{i_k}) = \prod_{j=1}^k \varphi(x_{i_j})$ .
- Further, extend  $\varphi$  to the ring  $\mathbb{F}\langle X_\infty \rangle$  by linearity:  $\varphi(\sum_{i=1}^t \alpha_i m_i) = \sum_{i=1}^t \alpha_i \varphi(m_i)$ , for monomials  $m_i \in X_\infty^*$  and scalars  $\alpha_i \in \mathbb{F}$  for  $i = 1$  to  $t$ .

► **Lemma 6.** *The map  $\varphi$  defined above is an injective homomorphism (i.e. a homomorphic embedding) from the ring  $\mathbb{F}\langle X_\infty \rangle$  to  $\mathbb{F}\langle x, y \rangle$ .*

**Proof.** To see that  $\varphi$  is a homomorphism, we first note that, by linearity, we have  $\varphi(f + g) = \varphi(f) + \varphi(g)$  for  $f, g \in \mathbb{F}\langle X_\infty \rangle$ . To verify that  $\varphi(fg) = \varphi(f)\varphi(g)$ , let  $f = \sum_m f_m m$  and  $g = \sum_m g_m m$  where  $f_m, g_m \in \mathbb{F}$  are the coefficients of monomial  $m$  in  $f$  and  $g$ , respectively. Then  $\varphi(fg) = \varphi((\sum_m f_m m)(\sum_w g_w w)) = \varphi(\sum_{m,w} f_m g_w m w)$ . Which, by linearity of  $\varphi$ , equals  $\sum_{m,w} f_m g_w \varphi(m)\varphi(w) = \varphi(f)\varphi(g)$ .

In order to show  $\varphi$  is injective, it suffices to show  $\varphi(f) \neq 0$  for  $f \neq 0$ . Suppose  $m \in \text{supp}(f)$ . Then  $\varphi(m) \neq 0$ , by the definition of  $\varphi$ . Hence, if  $m$  is the only monomial in  $\text{supp}(f)$  it follows that  $\varphi(f) \neq 0$ .

Otherwise, suppose  $m' \in \text{supp}(f)$  and  $m' \neq m$ . Let  $u$  be largest common prefix of  $m$  and  $m'$ . Then  $m = ux_i v$  and  $m' = ux_j w$ , for monomials  $u, v, w \in X_\infty^*$  and  $x_i \neq x_j$ . Noting that  $\varphi(x_i) = u_i + \bar{u}_i$  and  $\varphi(x_j) = u_j + \bar{u}_j$  we have  $\varphi(m) = \varphi(u)(u_i + \bar{u}_i)\varphi(v)$  and  $\varphi(m') = \varphi(u)(u_j + \bar{u}_j)\varphi(w)$ . From the definition of  $\varphi$ , clearly  $\varphi(u)$  is a homogeneous polynomial in  $\mathbb{F}\langle x, y \rangle$ . Let  $\deg(\varphi(u)) = D$ . Suppose  $\ell = |u_i| = |\bar{u}_i|$  and  $\ell' = |u_j| = |\bar{u}_j|$ . Without loss of generality suppose that  $u_i \prec u_j$ . Hence  $\ell \leq \ell'$ . As  $u_i$  and  $u_j$  are minimally balanced,  $u_i$  cannot be a prefix of  $u_j$ . Also, as  $u_i[1] = x$  and  $\bar{u}_j[1] = y$ ,  $u_i$  cannot be a prefix of  $\bar{u}_j$ . Therefore, for any monomials  $w_1 \in \text{supp}(\varphi(m))$  and  $w_2 \in \text{supp}(\varphi(m'))$ ,  $w_1$  and  $w_2$  will differ in the length  $\ell$  subword starting at location  $D + 1$ . It follows that  $\text{supp}(\varphi(m)) \cap \text{supp}(\varphi(m')) = \emptyset$ . Hence,  $\varphi(f) \neq 0$  implying that  $\varphi$  is injective. ◀

The subalgebra  $C$  has the important property that if  $f \in C$  then all factors of  $f$  are in  $C$  as well. In order to keep our presentation self-contained we include a complete proof with more details than are given in [5].

► **Theorem 7** (Bergman; [5, Chapter 4, Theorem 5.2]). *Let  $f \in C$ . For any factorization  $f = g \cdot h$  the polynomials  $g$  and  $h$  are in  $C$ .*

**Proof.** First we show that all monomials of  $g$  have the same imbalance. Likewise, all monomials of  $h$  have the same imbalance. Suppose  $a_{min}$  and  $a_{max}$  are the minimum and the maximum imbalances of monomials of  $g$ . Let  $b_{min}$  and  $b_{max}$  be the minimum and the maximum imbalance of monomials of  $h$ . Let  $m_{min}$  be a smallest monomial (with respect to  $\prec$ ) in  $\text{supp}(g)$  with imbalance  $a_{min}$ , and  $m_{max}$  be a largest monomial (with respect to  $\prec$ ) in  $\text{supp}(g)$  with imbalance  $a_{max}$ . Let  $w_{min}, w_{max}$  be monomials similarly defined for polynomial  $h$  corresponding to  $b_{min}$  and  $b_{max}$ . Now consider the product monomial  $u = m_{max}w_{max}$ . We claim that  $u$  is uniquely expressible as a product of a monomial of  $g$  and a monomial of  $h$ . To see this, suppose  $u = m'w'$  where  $m' \in \text{supp}(g)$ ,  $w' \in \text{supp}(h)$  and  $m_{max} \neq m'$  or  $w_{max} \neq w'$ . Now, as  $i(u) = i(m_{max}) + i(w_{max}) = i(m') + i(w')$  and  $m_{max}, w_{max}$  are the monomials with highest imbalance of  $g$  and  $h$  respectively, we must have  $i(m') = i(m_{max})$  and  $i(w') = i(w_{max})$ . So we get,  $m' \prec m_{max}$  and  $w' \prec w_{max}$  by the choice of  $m_{max}$  and  $w_{max}$ . But as  $u = m_{max}w_{max} = m'w'$ , clearly either  $m_{max}$  is a strict prefix of  $m'$  or  $w_{max}$  is a strict prefix of  $w'$ . In the former case we have  $m_{max} \prec m'$ , and in the later case  $w_{max} \prec w'$ .

These contradict the fact that  $m' \prec m_{max}$  and  $w' \prec w_{max}$ . Hence,  $u = m_{max}w_{max}$  is the unique expression of  $u$  as a product of a monomial of  $g$  and a monomial of  $h$ . Consequently,  $u$  has non-zero coefficient in  $f = g.h$ . Clearly  $u$  has imbalance  $a_{max} + b_{max}$ . Similarly, monomial  $v = m_{min}w_{min}$  is non-zero in  $f$  and has imbalance  $a_{min} + b_{min}$ . As  $f \in C$ , each monomial of  $f$  has imbalance 0. Hence,  $a_{max} + b_{max} = 0$  and  $a_{min} + b_{min} = 0$ . It follows that  $a_{max} = -b_{max} \leq -b_{min} = a_{min}$ , implying  $a_{min} = a_{max} = a$  and  $b_{min} = b_{max} = -a$ . Thus, all monomials of  $g$  have imbalance  $a$  and all monomials of  $h$  have imbalance  $-a$ .

Let  $m$  be the leading monomial of  $f$ . Clearly,  $m$  is a maximum degree monomial of  $f$ . Moreover,  $m$  is largest among the max-degree monomials of  $f$ . Let  $m = m_1m_2$  with  $m_1 \in \text{supp}(g)$  and  $m_2 \in \text{supp}(h)$ . We have  $i(m_1) = a$ ,  $i(m_2) = -a$ . As  $f \in C$ , the monomial  $\bar{m}$  obtained by replacing every occurrence of  $x$  by  $y$ , and  $y$  by  $x$  in  $m$  is also in  $\text{supp}(f)$ . Moreover,  $\bar{m}$  is the smallest monomial among the max-degree monomials of  $f$ . This forces that the monomial  $\bar{m}_1$  (obtained by interchanging  $x, y$  in  $m_1$ ) is in  $\text{supp}(g)$ . Similarly, monomial  $\bar{m}_2$  (obtained by swapping  $x, y$  in  $m_2$ ) is in  $\text{supp}(h)$ . We have  $i(\bar{m}_1) = -a$  and  $i(\bar{m}_2) = a$ . Now, all monomials of  $g$  have the same imbalance, and  $m_1, \bar{m}_1 \in \text{supp}(g)$ . This forces  $a = -a = 0$ . Consequently, all monomials in  $\text{supp}(g) \cup \text{supp}(h)$  have imbalance zero which implies  $g, h \in B$ . Now, applying Lemma 5 to  $g$  and  $h$  we have:

1.  $g = g_1 + g_2$ ,  $h = h_1 + h_2$ ,  $g_1, h_1 \in C$ , such that  $\text{lm}(g_2)$  has a  $\bar{T}$ -factor  $\bar{u}$ , and  $\text{lm}(h_2)$  has a  $\bar{T}$ -factor  $\bar{v}$ .
2. Consequently, the  $\text{deg}(g_2)$  prefix of  $\text{lm}(g_2h_1)$  contains the  $\bar{T}$ -factor  $\bar{u}$  and the  $\text{deg}(h_2)$  suffix of  $\text{lm}(g_1h_2)$  contains the  $\bar{T}$ -factor  $\bar{v}$ .
3. Finally, the  $\text{deg}(g_2)$  prefix and the  $\text{deg}(h_2)$  suffix of  $\text{lm}(g_2 \cdot h_2)$  contain, respectively, the  $\bar{T}$ -factors  $\bar{u}$  and  $\bar{v}$ .

Hence the leading monomials  $\text{lm}(g_2 \cdot h_1)$ ,  $\text{lm}(g_1 \cdot h_2)$ , and  $\text{lm}(g_2 \cdot h_2)$  are all distinct and cannot mutually cancel. Therefore, the leading monomial of  $\hat{f} = g_2 \cdot h_1 + g_1 \cdot h_2 + g_2 \cdot h_2$  contains a  $\bar{T}$ -factor unless both  $g_2 = 0$  and  $h_2 = 0$ . Now,  $\hat{f} = g_2 \cdot h_1 + g_1 \cdot h_2 + g_2 \cdot h_2 = f - g_1h_1$ . As  $f \in C$  and  $g_1, h_1 \in C$  it implies  $\hat{f} \in C$ . However, by Lemma 5, the leading monomial of  $\hat{f}$  cannot have a  $\bar{T}$ -factor. It forces  $g_2 = 0$  and  $h_2 = 0$  which implies  $g, h \in C$ . ◀

Theorem 7 implies that  $\varphi$  is a 1-inert embedding (Definition 2).

► **Theorem 8.** *Let  $f \in \mathbb{F}\langle X \rangle$ , where  $X = \{x_1, \dots, x_n\}$ . Suppose  $f' = \varphi(f) = g' \cdot h'$  is a non-trivial factorization of  $\varphi(f)$  in  $\mathbb{F}\langle x, y \rangle$ . Then there is a non-trivial factorization  $f = g \cdot h$  for  $g, h \in \mathbb{F}\langle X \rangle$ , such that  $\varphi(g) = g'$  and  $\varphi(h) = h'$ .*

**Proof.** As  $\mathbb{F}\langle X \rangle \subset \mathbb{F}\langle X_\infty \rangle$ , the embedding  $\varphi$  maps  $f \in \mathbb{F}\langle X \rangle$  to some  $f' = \varphi(f) \in C$ . Suppose  $f' = g' \cdot h'$  is a nontrivial factorization of  $f'$  in  $\mathbb{F}\langle x, y \rangle$ . By Theorem 7, as  $f' \in C$ , its factors  $g', h' \in C$ . Since  $g' \in C$ , it is an  $\mathbb{F}$ -linear combination of products of the form  $(u_{t_1} + \bar{u}_{t_1})(u_{t_2} + \bar{u}_{t_2}) \dots (u_{t_\ell} + \bar{u}_{t_\ell})$ . By definition of  $\varphi$ ,

$$(u_{t_1} + \bar{u}_{t_1})(u_{t_2} + \bar{u}_{t_2}) \dots (u_{t_\ell} + \bar{u}_{t_\ell}) = \varphi(x_{t_1}x_{t_2} \dots x_{t_\ell}).$$

Hence, by linearity, it follows that  $g' = \varphi(g)$  for some nontrivial polynomial  $g \in \mathbb{F}\langle X_\infty \rangle$ , similarly there is a nontrivial polynomial  $h \in \mathbb{F}\langle X_\infty \rangle$  such that  $h' = \varphi(h)$ . Since  $\varphi$  is a homomorphism, we have

$$\varphi(f) = f' = g' \cdot h' = \varphi(g) \cdot \varphi(h) = \varphi(g \cdot h).$$

As  $\varphi$  is injective, we have  $f = g \cdot h$ . To complete the proof we need to argue that  $g, h \in \mathbb{F}\langle X \rangle$ . Let  $\text{Var}(g)$  be the subset of variables that occur in some non-zero monomial of  $g$ . We claim that  $\text{Var}(g) \subseteq X$ . Suppose  $\text{Var}(g)$  contains some  $x_i \notin X$ . Let  $m \in \text{supp}(g)$  be the largest

monomial (in  $\prec$ -ordering) in which  $x_i$  occurs. Then the monomial  $m \cdot \text{lm}(h)$  contains the variable  $x_i$  and has a non-zero coefficient in  $f = gh$ . This is a contradiction as  $f \in \mathbb{F}\langle X \rangle$  and  $X$  does not contain  $x_i$ . Hence  $\text{Var}(g) \subseteq X$ . Similarly,  $\text{Var}(h) \subseteq X$ .  $\blacktriangleleft$

### 3 Multivariate to Bivariate reduction

We now apply Bergman's theorem (Theorem 7) to show that multivariate noncommutative polynomial factorization is reducible to bivariate noncommutative polynomial factorization. We require some preparatory observations.

Let  $X = \{x_1, x_2, \dots, x_n\}$ , and  $v_1, v_2, \dots, v_n$  be any  $n$  distinct and minimally balanced monomials in  $\{x, y\}^*$ . We define  $\varphi : \mathbb{F}\langle X \rangle \rightarrow \mathbb{F}\langle x, y \rangle$ :  $\varphi(x_i) = v_i + \bar{v}_i$  for all  $i$ , which extends by multiplication, i.e.  $\varphi(x_{i_1}x_{i_2}\dots x_{i_k}) = \prod_{j=1}^k \varphi(x_{i_j})$ , to monomials, and by linearity to  $\mathbb{F}\langle X \rangle$ . The definition of  $\varphi$  is essentially like in the proof of Bergman's theorem, except that  $X$  is finite and the  $v_i, 1 \leq i \leq n$  are any  $n$  distinct minimally balanced monomials. The following lemma is on the same lines as Theorem 7 and Theorem 8. The straightforward proof is by a suitable renaming of the variables  $x_1, \dots, x_n$  before and after application of Theorem 7 in the proof of Theorem 8.

► **Lemma 9.** *Let  $X = \{x_1, \dots, x_n\}$  and  $f \in \mathbb{F}\langle X \rangle$ . Suppose  $v_1, v_2, \dots, v_n \in \{x, y\}^*$  are distinct minimally balanced monomials. If  $f' = \varphi(f) = g' \cdot h'$  is a non-trivial factorization of  $f'$  in  $\mathbb{F}\langle x, y \rangle$  then there are polynomials  $g, h \in \mathbb{F}\langle X \rangle$  such that  $g' = \varphi(g)$ ,  $h' = \varphi(h)$  and  $f = g \cdot h$ .*

In order to obtain a polynomial-time computable reduction it is convenient to choose  $v_1, v_2, \dots, v_n$  such that each  $v_i$  has the same length. The next lemma ensures that  $\ell = O(\log n)$  suffices. This follows from the fact that the number of minimally balanced monomials of length  $2\ell$  is at least as large as the  $(\ell - 2)^{\text{th}}$  Catalan number, and well-known asymptotic lower bounds on Catalan numbers.

► **Lemma 10.** *There are at least  $n$  minimally balanced monomials of length  $2\ell$  in  $\{x, y\}^*$  for  $\ell \geq \max(\lceil \log 2n \rceil, 6)$ . Furthermore, the lexicographically first  $n$  minimally balanced monomials of length  $2\ell$  can be computed in time polynomial in  $n$ .*

**Proof.** Consider monomials  $v$  of the form  $v = x \cdot w \cdot y$ , where  $w$  is a Dyck monomial.<sup>3</sup> That is,  $w$  is a balanced monomial such that every prefix of  $w$  has at most as many  $y$ 's as  $x$ 's. Notice that  $w \in \{x, y\}^{2\ell-2}$ . It follows that any nontrivial prefix of  $v$  has strictly more  $x$  than  $y$ . So any such monomial is minimally balanced of length  $2\ell$ . The number of Dyck monomials of length  $2\ell - 2$  is  $C_{\ell-1}$  (the  $(\ell - 1)^{\text{th}}$  Catalan number). A standard estimate yields  $C_k \sim \frac{4^k}{k^{3/2}\sqrt{\pi}}$ , which implies that  $C_k$  is  $2^{\Omega(k)}$ . Specifically,  $C_k > 2^k$  for  $k \geq 5$ . If  $n < 2^{\ell-1}$  and  $\ell \geq 6$  then there are at least  $n$  minimally balanced monomials of length  $2\ell$ , for  $\ell = \max(\lceil \log 2n \rceil, 6)$ . Clearly, we can compute the  $v_i, 1 \leq i \leq n$  by enumeration in  $\text{poly}(n)$  time.  $\blacktriangleleft$

#### 3.1 White-box reduction

We first describe the reduction in the white-box case for input polynomial  $f \in \mathbb{F}\langle X \rangle$  given by a noncommutative arithmetic circuit.

<sup>3</sup> Essentially a balanced parenthesis string with  $x$  as left and  $y$  as right parenthesis, respectively.

► **Lemma 11.** *Let  $X = \{x_1, \dots, x_n\}$  and  $f \in \mathbb{F}\langle X \rangle$  be a noncommutative polynomial given by arithmetic circuit  $C$  of size  $s$ . Then there is a deterministic polynomial time algorithm that outputs an arithmetic circuit computing the polynomial  $\varphi(f) \in \mathbb{F}\langle x, y \rangle$ , where the minimally balanced monomials  $v_i, 1 \leq i \leq n$  defining the map  $\varphi$  are as described by Lemma 10.*

**Proof.** For  $1 \leq i \leq n$ , we note that the sum of two monomials  $v_i + \bar{v}_i$  can be computed by a noncommutative arithmetic formula  $F_i$  of size  $O(\log n)$ . Let  $C'$  be the arithmetic circuit obtained from circuit  $C$  by replacing input variable  $x_i$  with the formula  $F_i$ . Clearly,  $C'$  computes  $\varphi(f)$  and its size is polynomially bounded. ◀

► **Lemma 12.** *For  $f \in \mathbb{F}\langle X \rangle$  suppose  $\varphi(f) = f'_1 \cdot f'_2 \cdots f'_r$  is a complete factorization of  $\varphi(f)$  in  $\mathbb{F}\langle x, y \rangle$  into irreducible factors  $f'_i \in \mathbb{F}\langle x, y \rangle$ . Then there are irreducible polynomials  $f_1, f_2, \dots, f_r \in \mathbb{F}\langle X \rangle$  such that  $f = f_1 f_2 \dots f_r$  and  $\varphi(f_i) = f'_i$  for each  $i$ .*

**Proof.** It follows by repeated application of Lemma 9 that if  $\varphi(f) = f'_1 \cdot f'_2 \cdots f'_r$ , is a factorization into irreducible factors  $f'_i \in \mathbb{F}\langle x, y \rangle$ , then there are polynomials  $f_1, f_2, \dots, f_r \in \mathbb{F}\langle X \rangle$  such that  $f = f_1 f_2 \dots f_r$  and  $\varphi(f_i) = f'_i$  for each  $i$ . We claim each  $f_i$  is irreducible. For, if  $f_i = g \cdot h$  is a nontrivial factorization of  $f_i$  in  $\mathbb{F}\langle X \rangle$  then clearly  $f'_i = \varphi(f_i) = \varphi(g)\varphi(h)$  is a nontrivial factorization of  $f'_i$ , which contradicts its irreducibility. ◀

Suppose  $C'_i$  is an arithmetic circuit of size  $s'_i$  for  $f'_i$  for  $i \in [r]$ . We will construct a circuit of size  $\text{poly}(s'_i, n)$  for  $f_i$  efficiently for each  $i \in [r]$ , which is the crucial part of our multivariate to bivariate reduction. The next lemma describes the algorithm crucial to the white-box reduction.

► **Lemma 13.** *Given as input a noncommutative arithmetic circuit  $C$  for the polynomial  $\varphi(g) \in \mathbb{F}\langle x, y \rangle$ , where  $g \in \mathbb{F}\langle X \rangle$  is a degree  $d$  polynomial,  $X = \{x_1, x_2, \dots, x_n\}$ , there is a deterministic polynomial-time algorithm, running in time  $\text{poly}(d, \text{size}(C), n)$  that computes a noncommutative arithmetic circuit  $C'$  for the polynomial  $g$ . Furthermore, if  $\varphi(g)$  is given by an algebraic branching program then the algorithm computes an algebraic branching program for  $g$ .*

**Proof.** The proof is based on the idea of evaluating a noncommutative arithmetic circuit on an automaton (specifically, a substitution automaton) described in [4] (see e.g., for related applications [2],[3]).

Let  $g' = \varphi(g)$ . Let  $g = \sum_m \alpha_m m$  where  $m \in X^*$  and  $\alpha_m$  is the coefficient of  $m$  in  $g$ . As noted before, the map  $\varphi$  has the property that  $\text{supp}(\varphi(m)) \cap \text{supp}(\varphi(m')) = \emptyset$  for monomials  $m \neq m'$  in  $X^*$ . Moreover if  $m = x_{i_1} x_{i_2} \dots x_{i_\ell}$  has nonzero coefficient  $\alpha_m$  in  $g$  then  $g'$  has a monomial  $m' = v_{i_1} v_{i_2} \dots v_{i_\ell}$  with coefficient  $\alpha_m$ . Hence, to retrieve an arithmetic circuit for  $g$  from the given circuit  $C'$  for  $g'$  our aim is to carry out the following transformation of the polynomial  $g'$  given by the circuit  $C'$ :

- Get rid of the monomials of  $g'$  containing of all  $\bar{v}_j \in \bar{T}$  for  $j \in [n]$ .
- For each remaining monomial  $m'$  of  $g'$  substitute  $x_i$  wherever the monomial  $v_i$  occurs as substring in  $m'$  for  $i \in [n]$ .

We will accomplish this transformation by evaluating the circuit  $C'$  at suitably chosen matrix substitutions  $x \leftarrow M_x$  and  $y \leftarrow M_y$ , where  $M_x$  and  $M_y$  will be  $N \times N$  matrices for polynomially bounded  $N$ . The resulting evaluation  $C'(M_x, M_y)$  will be an  $N \times N$  matrix. A designated entry of this matrix will contain the polynomial  $g$ . Clearly, if we can efficiently compute the claimed matrices  $M_x$  and  $M_y$  it will yield an arithmetic circuit  $C$  for the polynomial  $g$ . These matrices  $M_x$  and  $M_y$  will be obtained as transition matrices of a substitution automaton that will carry out the above transformation steps on the polynomial  $g'$ .

A finite substitution automaton  $\mathcal{A}$  is a deterministic finite automata  $\mathcal{A}$  along with a substitution map  $\delta : Q \times \{x, y\} \rightarrow Q \times (X \cup \mathbb{F})$  where  $Q$  is a set of states and  $X = \{x_1, x_2, \dots, x_n\}$  are noncommuting variables. For  $i, j \in Q$ ,  $a \in \{x, y\}$ ,  $u \in X \cup \mathbb{F}$ , if  $\delta(i, a) = (j, u)$ , it means that when automata  $\mathcal{A}$  in state  $i$  reads  $a$ , it replaces  $a$  by  $u$  and transitions to state  $j$ . For each  $a \in \{x, y\}$  we can define a  $|Q| \times |Q|$  transition matrix  $M_a$  such that  $M_a(i, j) = u$  if  $\delta(i, a) = (j, u)$  and 0 otherwise.

With  $\delta$  we associate projections  $\delta_1 : Q \times \{x, y\} \rightarrow Q$  and  $\delta_2 : Q \times \{x, y\} \rightarrow X \cup \mathbb{F}$  defined as  $\delta_1(i, a) = j$  and  $\delta_2(i, a) = u$  if  $\delta(i, a) = (j, u)$ . The functions  $\delta_1$  and  $\delta_2$  extend naturally to monomials: For  $w \in \{x, y\}^*$ ,  $\delta_1(i, w) = j$  means the automaton  $\mathcal{A}$  goes from state  $i$  to  $j$  on reading  $w$ . Let  $\tilde{w}_\ell$  denotes length  $\ell$  prefix of  $w$  and  $w_\ell$  denotes  $\ell^{\text{th}}$  symbol of  $w$  from left.  $\delta_2(i, w) = p$  means  $p = \prod_{\ell=0}^{|w|-1} \delta_2(\delta_1(i, \tilde{w}_\ell), w_{\ell+1})$ . Note that  $\delta_2(i, w)$  has the form  $\beta \cdot w'$  where  $\beta \in \mathbb{F}$ ,  $w' \in X^*$ . For  $\alpha \in \mathbb{F}$  define  $\delta_2(i, \alpha \cdot w)$  as  $\alpha \cdot \delta_2(i, w)$ .

Let  $g'(x, y) = \sum_m \alpha_m m \in \mathbb{F}\langle x, y \rangle$ . Then, the  $(s, t)^{\text{th}}$  entry of the  $|Q| \times |Q|$  matrix  $g'(M_x, M_y)$  is a polynomial  $g \in \mathbb{F}\langle X \rangle$  such that  $g = \sum_{m \in W_t} \alpha_m \delta_2(s, m)$ , where  $W_t$  is the set of all monomials that take the automaton  $\mathcal{A}$  from state  $s$  to state  $t$ .

Clearly, if  $g'$  has an arithmetic circuit of size  $s$  then we can construct an arithmetic circuit of size  $\text{poly}(s, n, |Q|)$  for  $g$  in deterministic time  $\text{poly}(s, n, |Q|)$ .

Turning back to the reduction, consider the input circuit  $C$  for  $g' = \varphi(g) \in \mathbb{F}\langle x, y \rangle$ . We will construct a substitution automaton  $\mathcal{A}$  such that the polynomial  $g$  is the  $(s, t)^{\text{th}}$  entry of the matrix  $g'(M_x, M_y)$ .

### Description of the Substitution Automata

As each  $v_i$  is minimally balanced it must begin with symbol  $x$  and end with symbol  $y$ . As  $|v_i| > 2$ , the second symbol of  $v_i$  is also  $x$  (if it was  $y$ , then the balanced monomial  $xy$  would be a strict prefix of a minimally balanced monomial  $v_i$ , which is a contradiction). So clearly each  $v_i$  is of the form  $xxw_iyy$ , where  $w_i$  is a Dyck monomial. Let  $v'_i = xw_iy$  for  $i \in [n]$ . We can easily design a deterministic finite automaton  $A'$  with  $O(mn)$  states such that the language accepted by  $A'$  is precisely the finite set  $\{v'_1, v'_2, \dots, v'_n\}$ , where  $m$  is the length of  $v_i$  for  $i \in [n]$ . Let  $\delta'$  denote the transition function and  $Q'$  be the set of states of  $A'$ , where  $q_1$  is the initial state and  $q_{f_i}$  is the final state associated with acceptance of string  $v'_i$  for  $i \in [n]$ .  $A'$  has a tree structure with root  $q_1$  and leaves  $q_{f_i}$  for  $i \in [n]$ , and any root to leaf path has length exactly  $2\ell - 2$ . We now define the substitution automaton  $\mathcal{A}$ . Its state set is  $Q = Q' \cup \{q_0, q_f, q_r\}$ . The transition function  $\delta : Q \times \{x, y\} \rightarrow Q \times (X \cup \mathbb{F})$  is defined as follows:

1.  $\delta(q_0, x) = (q_1, 1); \delta(q_0, y) = (q_r, 0)$ .
2. for  $q \in Q' \setminus \{q_{f_i} | 1 \leq i \leq n\}$ . and  $a \in \{x, y\}$ , let  $\delta(q, a) = (\delta'(q, a), 1)$ .
3.  $\delta(q_{f_i}, x) = (q_r, 0); \delta(q_{f_i}, y) = (q_f, x_i)$  for each  $i \in [n]$ .
4.  $\delta(q_f, x) = (q_1, 1)$  and  $\delta(q_f, y) = (q_r, 0)$ .
5.  $\delta(q_r, a) = (q_r, 0)$  for  $a \in \{x, y\}$ .

The final state of  $\mathcal{A}$  is  $q_f$ . For a monomial  $w \in \{x, y\}^*$ , starting at state  $q_0$  the automaton  $\mathcal{A}$  substitutes all the variables with 1 as long as it matches with a prefix of  $v_i$  for  $i \in [n]$  (given by transitions in 1,2 above). When the monomial matches with  $v_i$  for some  $i$  (which will happen while reading symbol  $y$  as each string  $v_i$  ends with  $y$ ),  $\mathcal{A}$  substitutes  $y$  by  $x_i$  and moves to state  $q_f$ . If it reads  $x$  instead of  $y$  then  $\mathcal{A}$  enters a rejecting state  $q_r$  (given by transition in 3 above). Hence, if  $\mathcal{A}$  finds substring  $v_i$  in  $w$  it replaces it with  $x_i$ . Whenever  $\mathcal{A}$  is in state  $q_f$ , it means the monomial read so far is of the form  $v_{i_1}v_{i_2} \dots v_{i_t}$ , and it has replaced it with  $x_{i_1}x_{i_2} \dots x_{i_t}$ . If in the state  $q_f$  symbol  $y$  is encountered, it means the next

## 14:10 Multivariate to Bivariate Reduction for Noncommutative Polynomial Factorization

substring cannot match with a minimally balanced monomial (as these start with  $x$ ) and the automaton goes to the rejecting state  $q_r$ . If in state  $q_f$  variable  $x$  is read the automaton goes to state  $q_1$  and restarts the search for a new substring that matches with some  $v_i$  (transition in 4 above).

In conclusion  $\mathcal{A}$  replaces all the monomials of the form  $v_{i_1}v_{i_2}\dots v_{i_t}$  by  $x_{i_1}x_{i_2}\dots x_{i_t}$ . If the monomial contains an occurrence of  $\bar{v}_i$ , or it is not of the form  $v_{i_1}v_{i_2}\dots v_{i_t}$ , then  $\mathcal{A}$  zeros out that monomial by suitably setting an occurrence of  $y$  to zero or enters the reject state  $q_r$ .<sup>4</sup>

It follows that the  $(q_0, q_f)^{th}$  entry of the  $|Q| \times |Q|$  matrix  $g'(M_x, M_y)$  is the polynomial  $g$ , where  $g' = \varphi(g)$ , and  $M_x, M_y$  are the transition matrices for the substitution automaton  $\mathcal{A}$ . This completes the proof.

Finally, if  $\varphi(g)$  is given by an algebraic branching program  $P$  then it is easy to see that the above construction with the substitution automaton  $\mathcal{A}$  yields  $P(M_x, M_y)$  which is an algebraic branching program. ◀

The main theorem of this section, stated below, summarizes the discussion in this section.

► **Theorem 14.** *In the white-box setting, factorization of multivariate noncommutative polynomials into irreducible factors is deterministic polynomial-time reducible to factorization of bivariate noncommutative polynomials into irreducible factors. More precisely, given as input  $f \in \mathbb{F}\langle X \rangle$  by an arithmetic circuit (resp. algebraic branching program), the problem of computing a complete factorization  $f = f_1 \cdot f_2 \cdots f_r$  where each  $f_i$  is output as an arithmetic circuit (resp. algebraic branching program) is deterministic polynomial-time reducible to the same problem for bivariate polynomials in  $\mathbb{F}\langle x, y \rangle$ .*

**Proof.** We describe the reduction:

1. Input  $f \in \mathbb{F}\langle X \rangle$  (as a circuit or ABP).
2. Transform  $f$  to  $f' = \varphi(f) \in \mathbb{F}\langle x, y \rangle$  as a circuit (resp. ABP) by the algorithm of Lemma 10.
3. Compute a complete factorization of  $f' = f'_1 \cdot f'_2 \cdots f'_r$ , where each  $f'_i \in \mathbb{F}\langle x, y \rangle$  is irreducible and is computed as a circuit (resp. ABP).
4. Apply the algorithm of Lemma 13 to obtain a complete factorization of  $f = f_1 \cdot f_2 \cdots f_r$ , where each  $f_i$  is irreducible and is output as a circuit (resp. ABP).

The correctness of the reduction and its polynomial time bound follow from Lemmas 9, 10 and 13. ◀

► **Remark 15.** We note that in the case  $\mathbb{F}$  is the field  $\mathbb{Q}$  (of rationals), we need to take into account the bit complexity of the rational numbers involved and argue that the reduction is still polynomial time computable. The main point to note here is that the reduction guarantees the size of the factor  $f_i$  is polynomially bounded in the size of  $g_i, 1 \leq i \leq r$ , where the size of  $g_i$  includes the sizes of any rational numbers that might be involved in the description of the arithmetic circuit (or ABP) for  $g_i$ .

► **Remark 16.** We note here that the ring  $\mathbb{F}\langle X \rangle$  is not a unique factorization domain. That is, a polynomial  $f \in \mathbb{F}\langle X \rangle$  may have, in general, multiple factorizations into irreducibles [6]. A standard example is the polynomial  $x + xyx$  which factorizes as  $x(1 + yx)$  as well as  $(1 + xy)x$ , where  $x, y, 1 + yx, 1 + xy$  are irreducible. As the map  $\varphi$  is an injective homomorphism, there is a 1-1 correspondence between factorizations of  $\varphi(f)$  and factorizations of  $f$ . More specifically, our reduction takes as input any complete factorization  $\varphi(f) = f'_1 f'_2 \cdots f'_r$  and computes the corresponding complete factorization  $f = f_1 f_2 \cdots f_r$  of  $f$ .

<sup>4</sup> We can dispense with the reject state  $q_r$ , as suitably setting an occurrence of  $y$  to 0 would also suffice. We have transitions to the reject state  $q_r$  for exposition.

► **Remark 17.** We note that the embedding  $\varphi$  does not preserve sparsity<sup>5</sup> of the polynomial  $f$ . More precisely, if the sparsity of the  $n$ -variate degree  $d$  polynomial  $f$  is  $s$  then the sparsity of the bivariate polynomial  $\varphi(f)$  is  $O(2^d s)$ . Thus, using this embedding map we do not get a reduction from sparse  $n$ -variate degree  $d$  polynomial factorization to sparse bivariate polynomial factorization, where  $s, d$  are allowed to be part of the running time. This problem remains unanswered.

### 3.2 Black-box reduction

The reduction in the black-box case is essentially identical. The only point to note, which is easy to see, is that the analogue of Lemma 13 holds in the black-box setting. We state that below. We recall what a black-box means in the noncommutative setting.

► **Definition 18.** A noncommutative polynomial  $f \in \mathbb{F}\langle X \rangle$  given by black-box essentially means we can evaluate  $f$  at any matrix substitution  $x_i \leftarrow M_i$ ,  $M_i \in \mathbb{F}^{N \times N}$ , where the cost of each evaluation is the matrix dimension  $N$ .

In the black-box setting, suppose we have an efficient algorithm for bivariate noncommutative polynomial factorization of degree  $D$  polynomials  $g \in \mathbb{F}\langle x, y \rangle$ , where the algorithm takes a black-box for  $g$  and outputs black-boxes for the irreducible factors of some factorization of  $g$  in time  $\text{poly}(D)$ . Then, given a black-box for a degree  $D$   $n$ -variate polynomial  $f \in \mathbb{F}\langle X \rangle$  as input, we require that the reduction transforms it into a black-box of a bivariate polynomial  $g \in \mathbb{F}\langle x, y \rangle$ , and from the output black-boxes of  $g$ 's irreducible factors, the reduction has to efficiently recover black-boxes for the corresponding irreducible factors of  $f$ .

► **Lemma 19.** Given as input a black-box for the polynomial  $\varphi(g) \in \mathbb{F}\{x, y\}$ , where  $g \in \mathbb{F}\langle X \rangle$  is a degree  $d$  polynomial,  $X = \{x_1, x_2, \dots, x_n\}$ , with matrix substitutions for  $x$  and  $y$  computed in deterministic polynomial-time we can obtain a black-box for the polynomial  $g \in \mathbb{F}\langle X \rangle$ .

**Proof.** The proof of Lemma 13 already implies this because the matrices  $M_x$  and  $M_y$  described there do not require  $\varphi(g)$  to be given in white-box as circuit or ABP. Thus, the black-box for  $\varphi(g)$  yields a black-box for  $g$  by accessing the  $(q_0, q_f)^{\text{th}}$  entry of the matrix output  $\varphi(g)(M_x, M_y)$ . ◀

As a consequence we obtain the claimed reduction from multivariate factorization to bivariate factorization in the black-box setting as well.

► **Theorem 20.** The problem of computing a complete factorization of  $f \in \mathbb{F}\langle X \rangle$  given by black-box is deterministic polynomial-time reducible to the problem of black-box computation of a complete factorization of polynomials in  $\mathbb{F}\langle x, y \rangle$ .

**Proof.** Given a black-box for  $f$  we obtain a black-box for  $\varphi(f)$  applying Lemma 10. Then, given a complete factorization  $\varphi(f) = f'_1 \cdot f'_2 \cdots f'_r$  where each factor  $f'_i$  is output by a black-box for it, by Lemma 19 we can obtain black-boxes for each  $f_i$ . This yields a complete factorization  $f = f_1 \cdot f_2 \cdots f_r$  of  $f$  where the factors are given by black-box. ◀

<sup>5</sup> The sparsity of a polynomial  $f$  is the number of monomials in  $\text{supp}(f)$ .

#### 4 Factorizing $4 \times 4$ linear matrices over $\mathbb{Q}$

We have shown in Section 3 that multivariate noncommutative polynomial factorization is efficiently reducible to the bivariate case. Suppose  $f \in \mathbb{F}\langle x, y \rangle$  is a bivariate polynomial given by a formula of size  $s$ . Applying Higman linearization [6], as done in [1], we can transform the problem to the factorization of bivariate linear matrices  $A_0 + A_1x + A_2y$ , where the matrices have size bounded by  $2s$ . In [1] the problem of factorizing an  $n$ -variate polynomial  $f \in \mathbb{F}\langle X \rangle$  given by a formula was solved in two steps when  $\mathbb{F}$  is a finite field: (i) Transform  $f$  to a linear matrix  $L$  and factorize  $L$  into irreducible factors by reducing it to the common invariant subspace problem, and (ii) extract the factors of  $f$  from the factors of  $L$ . This approach fails for  $\mathbb{F} = \mathbb{Q}$  because the common invariant subspace problem for matrices over  $\mathbb{Q}$  is at least as hard as factoring square-free integers [12]. In this section, we show that linear matrix factorization over  $\mathbb{Q}$ , even for  $4 \times 4$  bivariate linear matrices, remains at least as hard as factoring square-free integers. Thus, efficient polynomial factorization over  $\mathbb{Q}$  remains elusive even for bivariate polynomials. Our proof is based on ideas from Ronyai's work [12].

Let  $\alpha, \beta \in \mathbb{Q}$  be nonzero rationals. The *generalized quaternion algebra*  $H(\alpha, \beta)$  is the 4-dimensional algebra over  $\mathbb{Q}$  generated by elements  $1, u, v, uv$  where the rules for multiplication in  $H(\alpha, \beta)$  are given by  $u^2 = \alpha$ ,  $v^2 = \beta$ , and  $uv = -vu$ . A *simple algebra*  $\mathcal{A}$  over a field  $\mathbb{F}$  is an algebra that has no nontrivial two-sided ideal. The *center*  $C$  of algebra  $\mathcal{A}$  is the subalgebra consisting of all elements of  $\mathcal{A}$  that commute with every element of  $\mathcal{A}$ . Furthermore, it follows from some general theory [11, Chapter 1.6] that:

► **Fact 21.** *For any nonzero  $\alpha, \beta \in \mathbb{Q}$ , the algebra  $H(\alpha, \beta)$  is a simple algebra with center  $\mathbb{Q}$ . The algebra  $H(\alpha, \beta)$  is either a division algebra (which means no zero divisors in it) or is isomorphic to the algebra of  $2 \times 2$  matrices over  $\mathbb{Q}$  (which means it has zero divisors).*

The 4-dimensional algebra  $H(\alpha, \beta)$  can be represented as an algebra of  $4 \times 4$  matrices over  $\mathbb{Q}$ , which is the *regular representation*. The matrix corresponding to 1 is  $I_4$ , and the

matrices corresponding to  $u$  and  $v$  are  $M_u = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \alpha & 0 \end{bmatrix}$  and  $M_v = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ \beta & 0 & 0 & 0 \\ 0 & -\beta & 0 & 0 \end{bmatrix}$ .

We next show that factorizing  $4 \times 4$  bivariate linear matrices is at least as hard as finding zero divisors in generalized quaternion algebras.

► **Theorem 22.** *Finding zero divisors in an input quaternion algebra  $H(\alpha, \beta)$  is polynomial-time reducible to factorizing  $4 \times 4$  bivariate linear matrices  $A_0 + A_1x + A_2y$ , where each scalar matrix  $A_i$  is in  $\mathcal{M}_4(\mathbb{Q})$ .*

**Proof.** Let  $H(\alpha, \beta)$  be the given generalized quaternion algebra. Then  $H(\alpha, \beta) = \{a_0 + a_1u + a_2v + a_3uv \mid a_i \in \mathbb{Q}\}$ , where  $u^2 = \alpha$ ,  $v^2 = \beta$ , and  $uv = -vu$  defines the algebra multiplication.

We now consider factorizations of the  $4 \times 4$  linear matrix  $I_4 + M_u x + M_v y$ .

▷ **Claim 23.** The linear matrix  $I_4 + M_u x + M_v y$  is irreducible if and only if the quaternion algebra  $H(\alpha, \beta)$  is a division algebra.

**Proof of Claim.** Suppose the linear matrix  $L = I_4 + M_u x + M_v y$  has a nontrivial factorization  $L = I_4 + M_u x + M_v y = FG$ . That means neither  $F$  nor  $G$  is a scalar matrix. By a theorem of Cohn [6, Theorem 5.8.8], there are invertible scalar matrices  $P$  and  $Q$  in  $\mathcal{M}_4(\mathbb{Q})$  such that

$$PLQ = \begin{bmatrix} A & 0 \\ D & B \end{bmatrix}.$$

► **Remark 24.** To apply Cohn’s theorem the matrix  $L$  needs to be monic. That is, the matrix  $[M_u \mid M_v]$  must have full row rank and  $[M_u^T \mid M_v^T]^T$  must have full column rank. This is ensured as matrices  $M_u$  and  $M_v$  are full rank.

Putting  $x = y = 0$  we observe that  $PQ = \begin{bmatrix} A_0 & 0 \\ D_0 & B_0 \end{bmatrix}$ , where  $A_0, B_0$  and  $D_0$  are scalar matrices. As  $P$  and  $Q$  are invertible, both  $A_0$  and  $B_0$  are invertible. Hence  $PLP^{-1} = \begin{bmatrix} A & 0 \\ D & B \end{bmatrix} \cdot \begin{bmatrix} A_0 & 0 \\ D_0 & B_0 \end{bmatrix}^{-1} = \begin{bmatrix} A' & 0 \\ D' & B' \end{bmatrix}$ , where  $A', B'$  and  $D'$  are also linear matrices. Recall that  $I_4, M_u$  and  $M_v$  are the matrix representations of the elements  $1, u,$  and  $v$  in the basis  $\{1, u, v, uv\}$  of  $H(\alpha, \beta)$ . Treating  $P$  as a basis change matrix, the above equation yields a new basis  $\{w_1, w_2, w_3, w_4\}$  of  $H(\alpha, \beta)$ . Let  $\dim(A') = k$ . Then  $1 \leq \dim(A') \leq 3$  and the vectors  $w_1, \dots, w_k$  spans a  $k$ -dimensional subspace  $W \subset H(\alpha, \beta)$  that is a common invariant subspace for the matrices  $I_4, M_u, M_v$  and  $M_{uv}$ . In other words, the subspace  $W$  is preserved under left multiplication by  $u$  and  $v$ . We can assume, without loss of generality, that  $w_1 \neq 1$ : if  $k > 1$  then clearly we can assume this. If  $k = 1$  notice that  $w_1 = 1$  is impossible because the subspace  $W$  is not preserved under left multiplication by  $u$  or  $v$ . Then the four elements  $w_1, uw_1, vw_1, uvw_1$  are all in  $W$  and hence linearly dependent. Thus, some nontrivial linear combination  $\gamma_0 w_1 + \gamma_1 u w_1 + \gamma_2 v w_1 + \gamma_3 uv w_1 = 0$ . which means  $(\gamma_0 + \gamma_1 u + \gamma_2 v + \gamma_3 uv) \times w_1 = 0$ . Hence  $w_1$  is a zero divisor in  $H(\alpha, \beta)$ . Conversely, if  $z \in H(\alpha, \beta)$  is a zero divisor then the left ideal  $J = \{xz \mid x \in H(\alpha, \beta)\}$  is a proper subspace of  $H(\alpha, \beta)$  that is invariant under  $M_u$  and  $M_v$ . Applying Cohn’s theorem [6, Theorem 5.8.8], we obtain invertible scalar matrices  $P$  and  $Q$  such that  $PLQ = \begin{bmatrix} A & 0 \\ D & B \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ D & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ 0 & B \end{bmatrix}$ . ◁

To complete the reduction, notice that if  $I_4 + M_u x + M_v y$  is irreducible then  $H(\alpha, \beta)$  is a division algebra. On the other hand, if we are given a nontrivial factorization  $I_4 + M_u x + M_v y = FG$  then, analyzing the proof of Cohn’s theorem [6, Theorem 5.8.8] (also see [1] for details), by suitable row and column operations we can compute in polynomial time the invertible scalar matrices  $P$  and  $Q$  from the factors  $F$  and  $G$ . Hence, by the proof of the above claim, we can efficiently compute a zero divisor  $w_1$  in  $H(\alpha, \beta)$ . ◀

As finding zero-divisors in the quaternion algebra  $H(\alpha, \beta)$  is known to be at least as hard as square-free integer factorization [12] we have the following.

► **Corollary 25.** *Factorizing  $4 \times 4$  bivariate linear matrices over  $\mathbb{Q}$  is at least as hard as factorizing square-free integers.*

## 5 Factorizing $3 \times 3$ linear matrices over $\mathbb{Q}$

In this section we present a deterministic polynomial-time algorithm for factorization of  $3 \times 3$  multivariate linear matrices over  $\mathbb{Q}$ . We start with a simple observation about linear matrix factorization in general.

► **Lemma 26.** *Suppose  $L = I_d + \sum_{i=1}^n A_i x_i$  is a linear matrix where each  $A_i, 0 \leq i \leq d$  is a  $d \times d$  matrix over  $\mathbb{Q}$ . Then  $L$  is irreducible if the characteristic polynomial of  $A_i$  is irreducible over  $\mathbb{Q}$  for any  $i$ .*

**Proof.** For if  $L$  is reducible then there is an invertible scalar matrix  $P$  such that  $PLP^{-1} = \begin{bmatrix} A & 0 \\ D & B \end{bmatrix}$ , which implies that  $PA_i P^{-1} = \begin{bmatrix} A'_i & 0 \\ D'_i & B'_i \end{bmatrix}$ , for scalar matrices  $A'_i, B'_i,$  and  $D'_i$ . Thus, the characteristic polynomial of  $A_i$  is the product of the characteristic polynomials of  $A'_i$  and  $B'_i$  which is a nontrivial factorization. ◀

The proof of the following theorem is based on linear algebra and Cohn's theorem [6, Theorem 5.8.8].

► **Theorem 27.** *There is a deterministic polynomial-time algorithm for factorization of  $3 \times 3$  multivariate linear matrices over  $\mathbb{Q}$ .*

**Proof.** We will first consider linear matrices of the form  $L = I_3 + \sum_{i=1}^n A_i x_i$ , where each  $A_i \in \mathcal{M}_3(\mathbb{Q})$  and the  $x_i$  are noncommuting variables. The algorithm computes a complete factorization of  $L$  into (at most three) irreducible linear matrix factors. By Cohn's theorem [6, Theorem 5.8.8], either  $L$  is irreducible or there is an invertible scalar matrix  $P$  such that  $PLP^{-1} = \begin{bmatrix} A & 0 \\ D & B \end{bmatrix}$ . Either  $A$  or  $B$  is a  $1 \times 1$  matrix. If  $A$  is a  $1 \times 1$  matrix then corresponding to it there is a 1-dimensional common invariant subspace spanned by a vector, say  $v$ , for the matrices  $A_i, 1 \leq i \leq n$ . More precisely, the row vector  $v^T$  is an eigenvector for each matrix  $A_i$ , and  $v^T A_i = \lambda_i v^T$  where  $\lambda_i \in \mathbb{Q}$  is the corresponding eigenvalue of matrix  $A_i$  for each  $i$ . Likewise, if  $B$  is a  $1 \times 1$  matrix then there is a corresponding 1-dimensional common invariant subspace spanned by a (column) vector  $u$  such that  $A_i u = \mu_i u$  for eigenvalues  $\mu_i$  of  $A_i$ . In either case, the common eigenspace is easy to compute from the characteristic polynomial of say  $A_1$  and then verifying that it is an eigenspace for the remaining  $A_i$  as well. This will yield the factorization  $PLP^{-1} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ D & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 \\ 0 & B \end{bmatrix}$ , where  $B$  is a  $2 \times 2$  linear matrix. The problem now reduces to factorizing the linear matrix  $B = I_2 + \sum_{i=1}^n B_i x_i$ , where  $B_i \in \mathcal{M}_2(\mathbb{Q})$ . A simple case analysis described below yields a polynomial-time algorithm for factorization of  $B$ .

1. If the characteristic polynomial of any  $B_i$  is irreducible over  $\mathbb{Q}$  then the linear matrix  $B$  is clearly irreducible.
2. If some  $B_i$  has two distinct eigenvalues  $\lambda \neq \lambda' \in \mathbb{Q}$  then the corresponding eigenspaces are 1-dimensional, spanned by their eigenvectors  $u \neq u'$ . Then either  $u$  or  $u'$  has to be an eigenvector for every  $B_j$  (otherwise  $B$  is irreducible), in which case we have a factorization of  $B$ .
3. Suppose each  $B_i$  has only one eigenvalue  $\lambda_i$ . Then, by linear algebra, after a basis change  $B_i$  is either of the form  $\begin{bmatrix} \lambda_i & 1 \\ 0 & \lambda_i \end{bmatrix}$  in which case the eigenspace is 1-dimensional with eigenvector  $(10)^T$ . We can check if this eigenspace is invariant for each  $B_j$  or not as before. Otherwise, after basis change each  $B_i = \begin{bmatrix} \lambda_i & 0 \\ 0 & \lambda_i \end{bmatrix} = \lambda_i I_2$  and the factorization is

$$B = \begin{bmatrix} 1 + \sum_{i=1}^n \lambda_i x_i & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 + \sum_{i=1}^n \lambda_i x_i \end{bmatrix}. \quad \blacktriangleleft$$

---

## References

- 1 Vikraman Arvind and Pushkar S. Joglekar. On efficient noncommutative polynomial factorization via higran linearization. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPICs*, pages 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CCC.2022.12.
- 2 Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial time identity testing for noncommutative circuits. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 831–841, 2017. doi:10.1145/3055399.3055442.

- 3 Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Comput. Complex.*, 19(4):521–558, 2010. doi:10.1007/s00037-010-0299-8.
- 4 Vikraman Arvind, Gaurav Rattan, and Pushkar S. Joglekar. On the complexity of noncommutative polynomial factorization. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 38–49, 2015. doi:10.1007/978-3-662-48054-0\_4.
- 5 P. M. Cohn. *Free Rings and their Relations*. London Mathematical Society Monographs. Academic Press, 1985.
- 6 P. M. Cohn. *Free Ideal Rings and Localization in General Rings*. New Mathematical Monographs. Cambridge University Press, 2006. doi:10.1017/CB09780511542794.
- 7 Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. Operator scaling: Theory and applications. *Found. Comput. Math.*, 20(2):223–290, 2020. doi:10.1007/s10208-019-09417-z.
- 8 Graham Higman. The units of group-rings. *Proceedings of the London Mathematical Society*, s2-46(1):231–248, 1940. doi:10.1112/plms/s2-46.1.231.
- 9 Erich Kaltofen. Factorization of polynomials given by straight-line programs. *Adv. Comput. Res.*, 5:375–412, 1989.
- 10 Erich Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990. doi:10.1016/S0747-7171(08)80015-6.
- 11 Richard S. Pierce. *Associative Algebras*. Graduate Texts in Mathematics. Springer, 1982.
- 12 Lajos Rónyai. Simple algebras are difficult. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 398–408. ACM, 1987. doi:10.1145/28395.28438.



# Entropic Risk for Turn-Based Stochastic Games

Christel Baier  

Technische Universität Dresden, Germany

Krishnendu Chatterjee  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Tobias Meggendorfer  

Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Technische Universität München, Germany

Jakob Piribauer  

Technische Universität Dresden, Germany

Technische Universität München, Germany

---

## Abstract

*Entropic risk (ERisk)* is an established risk measure in finance, quantifying risk by an exponential re-weighting of rewards. We study ERisk for the first time in the context of turn-based stochastic games with the total reward objective. This gives rise to an objective function that demands the control of systems in a risk-averse manner. We show that the resulting games are determined and, in particular, admit optimal memoryless deterministic strategies. This contrasts risk measures that previously have been considered in the special case of Markov decision processes and that require randomization and/or memory. We provide several results on the decidability and the computational complexity of the threshold problem, i.e. whether the optimal value of ERisk exceeds a given threshold. In the most general case, the problem is decidable subject to Shaniel’s conjecture. If all inputs are rational, the resulting threshold problem can be solved using algebraic numbers, leading to decidability via a polynomial-time reduction to the existential theory of the reals. Further restrictions on the encoding of the input allow the solution of the threshold problem in  $\text{NP} \cap \text{coNP}$ . Finally, an approximation algorithm for the optimal value of ERisk is provided.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic and verification

**Keywords and phrases** Stochastic games, risk-aware verification

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.15

**Related Version** *Full Version:* <https://arxiv.org/abs/2307.06611> [3]

**Funding** This work was partly funded by the ERC CoG 863818 (ForM-SMArt), the DFG Grant 389792660 as part of TRR 248 (Foundations of Perspicuous Software Systems), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany’s Excellence Strategy), and the DFG projects BA-1679/11-1 and BA-1679/12-1.

## 1 Introduction

**Stochastic Models.** Formal analysis of stochastic models is ubiquitous across disciplines of science, such as computer science [4], biology [43], epidemiology [29], and chemistry [28], to name a few. In computer science, a fundamental stochastic model are Markov decision processes (MDPs) [46], which extend purely stochastic Markov chains (MCs) with non-determinism to represent an agent interacting with a stochastic environment. Stochastic games (SGs) [49, 18, 19] in turn generalize MDPs by introducing an adversary, modelling the case where two agents engage in adversarial interaction in the presence of a stochastic environment. Notably, SGs can also be used to conservatively model MDPs where transition probabilities are not known precisely [15, 53]. See also [46] and [14, 21] for further applications of MDPs and SGs.



© Christel Baier, Krishnendu Chatterjee, Tobias Meggendorfer, and Jakob Piribauer; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

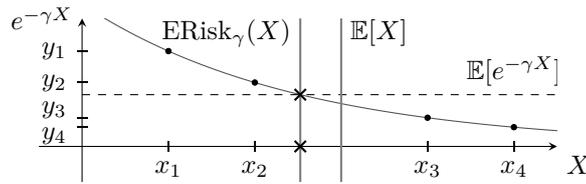
Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 15; pp. 15:1–15:16

Leibniz International Proceedings in Informatics



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

## 15:2 Entropic Risk for Turn-Based Stochastic Games



■ **Figure 1** Illustration of the entropic risk measure. The random variable  $X$  takes values  $x_1$  to  $x_4$  uniformly with probability  $\frac{1}{4}$  each. Expectation considers the average of  $x_i$ , while entropic risk yields the (normalized logarithm of the) average of  $y_i = e^{-\gamma x_i}$ .

**Strategies and Objectives.** In MDPs and SGs, the recipes to resolve choices are called strategies. The objective of the agent is to optimize a payoff function against all possible strategies of the adversary. One of the most fundamental problems studied in the context of MDPs and SGs is the optimization of total reward (and the related *stochastic shortest path* problem [7]). Here, every state (or, equivalently, transition) of the stochastic model is assigned a cost or reward and the payoff of a trajectory is the total sum of rewards appearing along the path. MDPs and SGs with total reward objectives provide an appropriate model to study a wide range of applications, such as traffic optimization [27], verification of stochastic systems [26, 47], or navigation / probabilistic planning [50].

**Risk-Ignorance of Expectation.** Typically, the expectation of the obtained total reward is optimized. However, the expectation measure is ignorant towards aspects of risk; an expectation maximizing agent accepts a one-in-a-million chance of extremely high rewards over a slightly worse, but guaranteed outcome. Such a behaviour might be undesirable in a lot of situations: Consider a one-shot lottery where with a chance of  $10^{-6}$  we win  $2 \cdot 10^6$  times our stake and otherwise lose everything – a two-times increase in expectation. The optimal strategy w.r.t. expectation would bet all available assets, ending up broke in nearly all outcomes.

**Risk-Aware Alternatives.** To address this issue, *risk-aware* objectives create incentives to prefer slightly smaller performance in terms of expectation in exchange for a more “stable” behaviour. To this end, several variants have been studied in the verification literature, such as (a) variance-penalized expected payoff [45, 22] that combines the expected value with a penalty for the variance of the resulting probability distribution; (b) trade-off of the expectation and variance for various notions of variance [40, 11]; (c) quantiles and conditional value-at-risk (CVaR) [47, 42, 35]; to name a few.

**Drawbacks.** The current approaches suffer from the following three drawbacks:

1. The above studies focus on the second moment (variance) along with the first moment (mean), but do not incorporate other moments of the payoff distribution.
2. All approaches are studied only for MDPs; none of them have been extended to SGs.
3. Even in MDPs, the above problems require complicated strategies. For example, trade-offs between expectation and variance require memory and randomization [11, 40], while optimizing variance-penalized expected payoffs, quantiles, or the CVaR of the total reward require exponential memory [45, 30, 44, 42].

**Entropic Risk.** The notion of entropic risk [24] has been widely studied in finance and operation research, see e.g. [23, 10]. Informally, instead of weighing each outcome uniformly and then aggregating it (as in the case for regular expectation), entropic risk re-weighs outcomes by an exponential function, then computes the expectation, and finally re-normalizes the value. We illustrate this in Figure 1. The exact definition of entropic risk is introduced later on.

**Advantages.** Aside from satisfying many desirable properties of risk measures established in finance, entropic risk brings several crucial advantages in our specific setting, of which we list a few: Compared to expectation, “bad” outcomes are penalized more than “good” outcomes add value. Thus, an agent optimizing entropic risk seeks to reduce the chances of particularly bad outcomes while also being interested in a good overall performance. In contrast to variance minimization, it is beneficial to increase the probability of extremely good outcomes (which would increase variance). Moreover, the entropic risk incorporates *all* moments of the distribution. In particular, even if the expectation is infinite, entropic risk still provides meaningful values (opposed to both expectation and variance). Note that the expected total reward objective is often addressed under additional assumptions excluding this case [8, 26]. Additionally, entropic risk is a *time-consistent* risk measure. In our situation, this means that the risk evaluation at a state is the same for *any history*. This is in stark contrast to, e.g., quantile and CVaR optimal strategies, which after a series of unfortunate events start behaving recklessly (e.g. expectation optimal). Due to these advantages, ERisk has already been studied in the context of MDPs [33, 5]. However, to the best of our knowledge, neither the arising computational problems nor the more general setting of SGs have been addressed.

## 1.1 Our results

In this work we consider the notion of entropic risk in the context of SGs as well as the special cases of MCs and MDPs. For an overview of our complexity results, see Table 1.

1. *Determinacy and Strategy Complexity.* We establish several basic results, in particular that SGs with the entropic risk objective are determined and that pure memoryless optimal strategies exist for both players. This stands in contrast to other notions of risk, where even in MDPs strategies require memory and/or randomization.
2. *Exact Computation.* When allowing Euler’s number  $e$  as the basis of exponentiation, the threshold problem whether the optimal entropic risk lies above a given bound is decidable subject to Shaniel’s conjecture. If the basis of exponentiation and all other numbers in the input are rational, then all numbers resulting from the involved exponentiation are shown to be algebraic. We obtain a reduction to the existential theory of the reals and thus a PSPACE upper bound in this case.

Furthermore, we identify a notion of *small algebraic instance* in which all occurring numbers are not only algebraic, but have a small representation and are contained in an algebraic extensions of  $\mathbb{Q}$  of low degree. The threshold problem for small algebraic instances of MCs and MDPs can efficiently be solved by explicit computations in an algebraic extension of  $\mathbb{Q}$ . We obtain polynomial-time algorithms for MCs and MDPs, and conclude that the threshold problem lies in  $\text{NP} \cap \text{co-NP}$  for SGs in this case. For small algebraic instances, we furthermore show that an explicit closed form of the optimal value can be computed (a) in polynomial time for MCs; and consequently (b) in polynomial space for SGs.

3. *Approximate Computation.* We provide an effective way to compute an approximation, i.e. determine the optimal entropic risk up to a given precision of  $\varepsilon > 0$ . To this end, we show that in the general case, by considering enough bits of arising irrational numbers, we can

■ **Table 1** Overview of the decidability and complexity results for SGs, MDPs and MCs.

	threshold problem			optimal value	
	general instances (Thm. 14)	algebraic instances (Thm. 16)	small algebraic instances (Thm. 19)	computation for small algebraic instances (Thm. 20)	approximation with small rewards and risk aversion factor (Thm. 21)
SGs	decidable subject to Shanuel's conjecture	in PSPACE (in $\exists\mathbb{R}$ )	in $\text{NP} \cap \text{coNP}$	in polynomial space	in polynomial space
MDPs			in PTIME		in polynomial time
MCs					

bound the incurred error. In MDPs and MCs, the optimal value can be approximated in time polynomial in the size of the model, in  $-\log(\varepsilon)$ , and in the magnitude of the rewards. For SGs, this implies the existence of a polynomial-space approximation algorithm.

## 1.2 Related Work

The entropic risk objective has been studied before in MDPs: An early formulation can be found in [33] under the name *risk-sensitive MDPs* focusing on the finite-horizon setting. The paper [34] considers an exponential utility function applied to discounted rewards and optimal strategies are shown to exist, but not to be memoryless in general. In [20], the entropic risk objective is considered for MDPs with a general Borel state space and in [5] a generalization of this objective is studied on such MDPs. To the best of our knowledge, however, all previous work in the context of MDPs focuses on optimality equations and general convergence results of value iteration, while the resulting algorithmic problems for finite-state MDPs have not been investigated. Furthermore, we are not aware of work on the entropic risk objective in SGs.

For other objectives capturing risk-aversion, algorithmic problems have been analyzed on finite-state MDPs: Variance-penalized expectation has been studied for finite-horizon MDPs with terminal rewards in [17] and for infinite-horizon MDPs with discounted rewards and mean payoffs [22], and total rewards [45]. For total rewards, optimal strategies require exponential memory and the threshold problem is in NEXPTIME and EXPTIME-hard [45].

In [40], the optimization of expected accumulated rewards under constraints on the variance are studied for finite-horizon MDPs. Possible tradeoffs between expected value and variance of mean payoffs and other notions of variability have been studied in [11].

To control the chance of bad outcomes, the problem to maximize or minimize the probability that the accumulated weight lies below a given bound  $w$  has been addressed in MDPs [30, 31]. Similarly, quantile queries ask for the minimal weight  $w$  such that the weight of a path stays below  $w$  with probability at least  $p$  for the given value  $p$  under some or all schedulers [51, 48]. Both of these problems have been addressed for MDPs with non-negative weights and are solvable in exponential time in this setting [51, 30]. Optimal strategies require exponential memory and the decision version of these problems is PSPACE-hard [30].

The conditional value-at-risk (CVaR), a prominent risk-measure, has been investigated for mean payoff and weighted reachability in MDPs in [35] as well as for total rewards in MDPs [44, 42]. The optimal CVaR of the total reward in MDPs with non-negative weights can be computed in exponential time and optimal strategies require exponential memory [44, 42]. The threshold problem for optimal CVaR of total reward in MDPs with integer weights is at least as hard as the Positivity-problem for linear recurrence sequences, a well-known problem in analytic number theory whose decidability status is, since many decades, open [44].

For all these objectives capturing risk-aversion in some sense, we are not aware of any work addressing the resulting algorithmic problems on SGs.

## 2 Preliminaries

In this section, we recall the basics of (turn-based) SGs and relevant objectives. For further details, see, e.g., [46, 4, 26, 21]. We assume familiarity with basic notions of probability theory (see, e.g., [9]). We write  $\mathcal{D}(X)$  to denote the set of all *probability distributions* over a countable set  $X$ , i.e. mappings  $d : X \rightarrow [0, 1]$  such that  $\sum_{x \in X} d(x) = 1$ . The support of a distribution  $d$  is  $\text{supp}(d) := \{x \in X \mid d(x) > 0\}$ . For a set  $S$ ,  $S^*$  and  $S^\omega$  refer to the set of finite and infinite sequences of elements of  $S$ , respectively.

### Markov Chains, MDPs, and Stochastic Games

A *Markov chain (MC)* (e.g. [4]), is a tuple  $M = (S, \delta)$ , where  $S$  is a set of *states*, and  $\delta : S \rightarrow \mathcal{D}(S)$  is a *transition function* that for each state  $s$  yields a probability distribution over successor states. We write  $\delta(s, s')$  instead of  $\delta(s)(s')$  for the probability to move from  $s$  to  $s'$  for  $s, s' \in S$ . A (*infinite*) *path* in an MC is an infinite sequence  $s_0, s_1, \dots$  of states such that for all  $i$ , we have  $\delta(s_i, s_{i+1}) > 0$ . We denote the set of infinite paths by  $\text{Paths}_M$ . Together with a state  $s$ , an MC  $M$  induces a unique probability distribution  $\text{Pr}_{M,s}$  over the set of all infinite paths  $\text{Paths}_M$  starting in  $s$ . For a random variable  $f : \text{Paths}_M \rightarrow \mathbb{R}$ , we write  $\mathbb{E}_{M,s}(f)$  for the expected value of  $f$  under the probability measure  $\text{Pr}_{M,s}$ .

A *turn-based stochastic game (SG)* (e.g. [18]) is a tuple  $(S_{\max}, S_{\min}, A, \Delta)$ , where  $S_{\max}$  and  $S_{\min}$  are disjoint sets of *Maximizer* and *Minimizer* states, inducing the set of states  $S = S_{\max} \cup S_{\min}$ ,  $A$  denotes a finite set of *actions*, furthermore overloading  $A$  to also act as a function assigning to each state  $s$  a set of non-empty *available actions*  $A(s) \subseteq A$ , and  $\Delta : S \times A \rightarrow \mathcal{D}(S)$  is the *transition function* that for each state  $s$  and (available) action  $a \in A(s)$  yields a distribution over successor states. For convenience, we write  $\Delta(s, a, s')$  instead of  $\Delta(s, a)(s')$ . Moreover,  $\text{opt}_{a \in A(s)}^s$  refers to  $\max_{a \in A(s)}$  if  $s \in S_{\max}$  and  $\min_{a \in A(s)}$  if  $s \in S_{\min}$ , i.e. the preference of either player in a state  $s$ . We omit the superscript  $s$  where clear from context. Given a function  $f : S \rightarrow \mathbb{R}$  assigning values to states, we write  $\Delta(s, a)\langle f \rangle := \sum_{s' \in S} \Delta(s, a, s') \cdot f(s')$  for the weighted sum over the successors of  $s$  under  $a \in A(s)$ . A *Markov decision process (MDP)* (e.g. [46]) can be seen as an SG with only one player, i.e.  $S_{\max} = \emptyset$  or  $S_{\min} = \emptyset$ .

The semantics of SGs is given in terms of resolving choices by strategies inducing an MC with the respective probability space over infinite paths. Intuitively, a stochastic game is played in turns: In every state  $s$ , the player to whom it belongs chooses an action  $a$  from the set of available actions  $A(s)$  and the play advances to a successor state  $s'$  according to the probability distribution given by  $\Delta(s, a)$ . Starting in a state  $s_0$  and repeating this process indefinitely yields an infinite sequence  $\rho = s_0 a_0 s_1 a_1 \dots \in (S \times A)^\omega$  such that for every  $i \in \mathbb{N}_0$  we have  $a_i \in A(s_i)$  and  $\Delta(s_i, a_i, s_{i+1}) > 0$ . We refer to such sequences as (*infinite*) *paths* or *plays* and denote the set of all infinite paths in a given game  $G$  by  $\text{Paths}_G$ . Furthermore, we write  $\rho_i$  to denote the  $i$ -th state in the path  $\rho$ . *Finite paths* or *histories*  $\text{FPaths}_G$  are finite prefixes of a play, i.e. elements of  $(S \times A)^* \times S$  consistent with  $A$  and  $\Delta$ .

The decision-making of the players is captured by the notion of *strategies*. Strategies are functions mapping a given history to a distribution over the actions available in the current state. For this paper, *memoryless deterministic* strategies (abbreviated *MD strategies*, also called positional strategies) are of particular interest. These strategies choose a single action in each state, irrespective of the history, and can be identified with functions  $\sigma : S \rightarrow A$ . Since we show that these strategies are sufficient for the discussed notions, we define the semantics of games only for these strategies and refer the interested reader to the mentioned

## 15:6 Entropic Risk for Turn-Based Stochastic Games

literature for further details. We write  $\Pi_G$  for the set of all strategies and  $\Pi_G^{\text{MD}}$  for memoryless deterministic ones. We call a pair of strategies a *strategy profile*, written  $\pi = (\sigma, \tau)$ . We identify a profile with the induced joint strategy  $\pi(s) := \sigma(s)$  if  $s \in S_{\max}$  and  $\tau(s)$  otherwise.

Given a profile  $\pi = (\sigma, \tau)$  of MD strategies for a game  $G$ , we write  $G^\pi$  for the MC obtained by fixing both strategies. So,  $G^\pi = (S, \hat{\delta})$ , where  $\hat{\delta}(s) := \Delta(s, \pi(s))$ . Together with a state  $s$ , the MC  $G^\pi$  induces a unique probability distribution  $\Pr_{G,s}^\pi$  over the set of all infinite paths  $\text{Paths}_G$ . For a random variable over paths  $f : \text{Paths}_G \rightarrow \mathbb{R}$ , we write  $\mathbb{E}_{G,s}^\pi[f]$  for the expected value of  $f$  under the probability measure  $\Pr_{G,s}^\pi$ .

### Objectives

Usually, we are interested in finding strategies that optimize the value obtained for a particular *objective*. We introduce some objectives of interest.

**Reachability.** A reachability objective is specified by a set of *target states*  $T \subseteq S$ . We define  $\diamond T = \{\rho \mid \exists i. \rho_i \in T\}$  the set of all paths eventually reaching a target state. Given a strategy profile  $\pi$  and a state  $s$ , the probability for this event is given by  $\Pr_{G,s}^\pi[\diamond T]$ . On games, we are interested in determining the *value*  $\text{Val}_{G, \diamond T}(s) := \max_{\sigma \in \Pi_G^{\text{MD}}} \min_{\tau \in \Pi_G^{\text{MD}}} \Pr_{G,s}^{\sigma, \tau}[\diamond T]$  of a state  $s$ , which intuitively is the best probability we can ensure against an optimal opponent. Generally, one would consider supremum and infimum over strategies instead maximum and minimum over MD strategies. However, for reachability we know that these value coincide and the game is *determined*, i.e. the order of max and min does not matter [19]. Finally, we know that the value  $\text{Val}_{G, \diamond T}$  is a solution of the following set of equations

$$v(s) = 0 \text{ for } s \in S_0, \quad v(s) = 1 \text{ for } s \in T, \quad \text{and } v(s) = \text{opt}_{a \in A(s)} \Delta(s, a) \langle v \rangle \text{ otherwise, (1)}$$

where  $S_0$  is the set of states that cannot reach  $T$  against an optimal Minimizer strategy [13].

**Total Reward.** The total reward objective is specified by a reward function  $r : S \rightarrow \mathbb{R}_{\geq 0}$ , assigning non-negative rewards to every state. The total reward obtained by a particular path is defined as the sum of all rewards seen along this path,  $\text{TR}(\rho) := \sum_{i=1}^{\infty} r(\rho_i)$ . Note that since we assume  $r(s) \geq 0$ , this sum is always well-defined. Classically, we want to optimize the expected total reward, i.e. determine  $\text{Val}_{G, \mathbb{E} \text{TR}}(s) := \max_{\sigma \in \Pi_G^{\text{MD}}} \min_{\tau \in \Pi_G^{\text{MD}}} \mathbb{E}_{G,s}^{\sigma, \tau}[\text{TR}]$ . This game is determined and MD strategies suffice [16]. (To be precise, that work considers a more general formulation of total reward, our case is equivalent to the case  $\star = c$  and  $T = \emptyset$  (Def. 3) and the quantitative rPATL formula  $\langle\langle\{1\}\rangle\rangle \mathbf{R}_{\max=?}^r[\mathbf{F}^c \mathbf{f} \mathbf{f}]$ .)

### 3 Entropic Risk

As hinted in the introduction, for classical total reward we optimize the expectation and disregard other properties of the actual distribution of obtained rewards. This means that an optimal strategy may accept arbitrary risks if they yield minimal improvements in terms of expectation. To overcome this downside, we consider the entropic risk:

► **Definition 1.** Let  $b > 1$  a basis,  $X$  a random variable, and  $\gamma > 0$  a risk aversion factor. The entropic risk (of  $X$  with base  $b$  and factor  $\gamma$ ) (see, e.g., [25]) is defined as

$$\text{ERisk}_\gamma(X) := -\frac{1}{\gamma} \log_b(\mathbb{E}[b^{-\gamma X}]).$$

One often chooses  $b = e$ . Nevertheless, we also consider rational values for  $b$ , which allows us to apply techniques from algebraic number theory to arising computational problems.

► **Example 2.** Consider a random variable  $X$  that takes values  $x_1 = 1$ ,  $x_2 = 2$ ,  $x_3 = 4$ , and  $x_4 = 5$  with probability  $1/4$  each. Figure 1 illustrates how the entropic risk measure of  $X$  with base  $e$  is obtained for some risk aversion factor  $\gamma$ : The values  $x_i$  are depicted on the  $x$ -axis. We now map the values  $x_i$  to values  $y_i = e^{-\gamma x_i}$  on the  $y$ -axis. Then, the expected value of  $e^{-\gamma X}$  can be obtained as the arithmetic mean of the values  $y_i$ . The result is mapped back to the  $x$ -axis via  $y \mapsto -\frac{1}{\gamma} \log(y)$ , the inverse of  $x \mapsto e^{-\gamma x}$ , and we obtain  $\text{ERisk}_\gamma(X)$ .

The example shows that deviations to lower values are penalized, i.e. taken into consideration more strongly, by this risk measure. For a different perspective, we can also consider the Taylor expansion of  $\text{ERisk}$  w.r.t.  $\gamma$ , which is  $\text{ERisk}_\gamma(X) = \mathbb{E}[X] - \frac{\gamma}{2} \cdot \text{Var}[X] + \mathcal{O}(\gamma^2)$  (see, e.g., [2]). The terms hidden in  $\mathcal{O}(\gamma^2)$  comprise all moments of  $X$  and exhibit an asymmetry such that  $\text{ERisk}$  is roughly the expected value minus a penalty for deviations to lower values.

### 3.1 Entropic Risk in SGs

We are interested in the case  $X = \text{TR}$ , i.e. optimizing the risk for total rewards. We write

$$\text{ERisk}_{\mathbb{G}, \hat{s}}^\gamma(\pi) := -\frac{1}{\gamma} \log_b(\mathbb{E}_{\mathbb{G}, \hat{s}}^\pi[b^{-\gamma X}])$$

to denote the entropic risk of the total reward achieved by the strategy profile  $\pi$  when starting in state  $\hat{s}$ , omitting sub- and superscripts where clear from context. Clearly, this is well defined for any profile: We have that  $b^{-\gamma \text{TR}(\rho)} = b^{-\gamma \sum_{i=1}^{\infty} r(\rho_i)} = \prod_{i=1}^{\infty} b^{-\gamma r(\rho_i)}$  and each factor lies between 0 and 1, thus the product converges (possibly with limit 0).

We also give an insightful characterization for integer rewards. If  $r(s) \in \mathbb{N}$ , we have

$$\text{ERisk}_{\mathbb{G}, \hat{s}}^\gamma(\pi) = -\frac{1}{\gamma} \log_b \left( \sum_{n=0}^{\infty} \Pr_{\mathbb{G}, \hat{s}}^\pi[\text{TR} = n] \cdot b^{-\gamma n} \right). \quad (2)$$

Naturally, our goal is to optimize the entropic risk. In this work, we mainly consider the corresponding decision variant, which we call the *entropic risk threshold problem*:

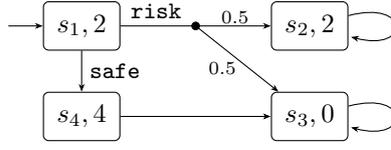
**Entropic risk threshold problem:** Given an SG  $\mathbb{G}$ , state  $\hat{s}$ , reward function  $r$ , risk parameter  $\gamma$ , risk basis  $b$ , and threshold  $t$ , decide whether there exists a Maximizer strategy  $\sigma$  such that for all Minimizer strategies  $\tau$  we have  $\text{ERisk}_{\mathbb{G}, \hat{s}}^\gamma((\sigma, \tau)) \geq t$ .

Note that (for now) we do not assume any particular encoding of the input. For example, the reward function  $r$  could be given symbolically, describing irrational numbers. A second variant of the threshold problem asks whether the optimal value

$$\text{ERisk}_{\mathbb{G}, \hat{s}}^{\gamma*} := \sup_{\sigma \in \Pi_{\mathbb{G}}} \inf_{\tau \in \Pi_{\mathbb{G}}} \text{ERisk}_{\mathbb{G}, \hat{s}}^\gamma((\sigma, \tau)) \quad (3)$$

is at least  $t$  for a given threshold  $t$ . We will see that SGs with the entropic risk as objective function are determined and hence the two variants are equivalent. Before proceeding with our solution approaches, we provide an illustrative example.

► **Example 3.** Consider the MDP of Figure 2. The optimal total reward is obtained by choosing action **risk** in state  $s_1$ : Then, we actually obtain an infinite total reward through state  $s_2$ . In comparison, choosing action **safe** would yield a reward of 6 in total. Now, consider the entropic risk. When choosing action **risk**, we obtain a total reward of 2 and  $\infty$  with probability  $\frac{1}{2}$  each, while action **safe** yields 6 with probability 1. Let  $b = 2$  and  $\gamma = 1$  for simplicity. Then, we obtain an entropic risk of  $-\log_2(\frac{1}{2}2^{-2} + \frac{1}{2}2^{-\infty}) = 3$  under action **risk** and  $-\log_2(2^{-6}) = 6$  for **safe**. Thus, action **safe** is preferable.



■ **Figure 2** Our running example to demonstrate several properties of entropic risk. For ease of presentation, the system actually is an MDP, where all states belong to Maximizer. States are denoted by boxes and their reward is written next to the state name. Transition probabilities are written next to the corresponding edges, omitting probability 1.

► **Remark 4.** As hinted above, entropic risk is finite whenever a finite reward is obtained with non-zero probability, i.e. for any strategy profile  $\pi$ ,  $\text{ERisk}_{\mathcal{G},s}^\gamma(\pi) = \infty$  iff  $\Pr_{\mathcal{G},s}^\pi[\text{TR} = \infty] = 1$ . In contrast, expectation is infinite whenever there is a non-zero chance of infinite reward, i.e.  $\mathbb{E}_{\mathcal{G},s}^\pi[\text{TR}] = \infty$  iff  $\Pr_{\mathcal{G},s}^\pi[\text{TR} = \infty] > 0$ . So, entropic risk allows us to meaningfully compare strategies which yield infinite total reward with some positive probability.

### 3.2 Exponential Utility

Observe that the essential part of the entropic risk is the inner expectation. Thus, we consider the *negative exponential utility*

$$\text{NegUtil}_{\mathcal{G},s}^\gamma(\pi) := \mathbb{E}_{\mathcal{G},s}^\pi[b^{-\gamma \text{TR}}].$$

We have  $\text{ERisk}_{\mathcal{G},s}^\gamma(\pi) = -\frac{1}{\gamma} \log_b(\text{NegUtil}_{\mathcal{G},s}^\gamma(\pi))$ . Observe that in our case  $0 \leq \text{NegUtil}_{\mathcal{G},s}^\gamma(\pi) \leq 1$  for any  $\pi$ , as  $0 \leq \text{TR} \leq \infty$ . Moreover,  $\text{ERisk}_{\mathcal{G},s}^\gamma(\pi) \geq t$  iff  $\text{NegUtil}_{\mathcal{G},s}^\gamma(\pi) \leq b^{-\gamma t}$ , thus, a risk-averse agent (in our case Maximizer) wants to minimize NegUtil. The optimal value is

$$\text{NegUtil}_{\mathcal{G},s}^{\gamma*} := \inf_{\sigma \in \Pi_{\mathcal{G}}} \sup_{\tau \in \Pi_{\mathcal{G}}} \mathbb{E}_{\mathcal{G},s}^{\sigma,\tau}[b^{-\gamma \text{TR}}]. \quad (4)$$

We again omit sub- and superscripts where clear from context. We show later that games with NegUtil or ERisk as payoff functions are determined. Thus, the order of sup and inf in the above definition does not matter. We call a Maximizer-strategy  $\sigma$  optimal if  $\text{ERisk}_{\mathcal{G},s}^{\gamma*} = \inf_{\tau \in \Pi_{\mathcal{G}}} \text{ERisk}_{\mathcal{G},s}^\gamma((\sigma, \tau))$  and analogously for Minimizer-strategies.

## 4 Basic Properties and Decidability

In this section, we establish several results for SGs with entropic risk as objective functions concerning determinacy, strategy complexity, and decidability in the general case. We mainly work on games with NegUtil as payoff function. As ERisk can be obtained from NegUtil via the monotone function  $-\frac{1}{\gamma} \log(\cdot)$ , most results, such as determinacy or strategy complexity, will transfer directly to games with ERisk as objective function.

First, we show that the games are determined, i.e. the order of sup and inf in Equation (3) and Equation (4) can be switched. Then, we show that games with NegUtil as payoff function can be seen as reachability games via a reduction that introduces irrational transition probabilities in general. We conclude that considering only MD strategies is sufficient to obtain the optimal value, i.e. sup and inf can be replaced with a max and min over MD strategies. From this, we derive a system of inequalities that has a solution if and only if the optimal value satisfies  $\text{ERisk}^* \geq t$  for a given threshold  $t$ . We conclude this section by observing that the satisfiability of this system of inequalities can be expressed as a sentence in the language of the reals with exponentiation. In this way, we obtain the conditional decidability of the entropic risk threshold problem in SGs subject to Shaniel's conjecture.

Throughout this section, fix a game  $G$ , reward function  $r$ , state  $\hat{s}$ , risk parameter  $\gamma$ , and risk basis  $b$ . Omitted proofs can be found in the extended version [3].

## 4.1 Determinacy and Optimality Equation

► **Lemma 5.** *Stochastic games with NegUtil as payoff function are determined, i.e.*

$$\inf_{\sigma \in \Pi_G} \sup_{\tau \in \Pi_G} \mathbb{E}_{G,s}^{\sigma,\tau}[b^{-\gamma \text{TR}}] = \sup_{\tau \in \Pi_G} \inf_{\sigma \in \Pi_G} \mathbb{E}_{G,s}^{\sigma,\tau}[b^{-\gamma \text{TR}}].$$

**Proof.** This follows from the classical result on determinacy of Borel games [41], see [39] for a concrete formulation for stochastic games. In particular, the game is zero-sum and NegUtil is a bounded, Borel-measurable function. ◀

As ERisk is obtained from NegUtil via a monotone function, also games with ERisk as payoff function are determined. While ERisk\* is difficult to tackle directly due to its non-linearity, we can derive the following optimality equation for NegUtil\*:

► **Lemma 6.** *The optimal utility NegUtil\* is a solution of the following system of constraints:*

$$v(s) = b^{-\gamma r(s)} \cdot \overline{\text{opt}}_{a \in A(s)}^s \cdot \sum_{s' \in S} \Delta(s, a, s') \cdot v(s'), \quad (5)$$

where  $\overline{\text{opt}}^s$  is min for a Maximizer state  $s$  and max for a Minimizer state.

Unfortunately, NegUtil is not the unique or, at least, the pointwise smallest or largest fixed point of this equation system. Consider the case where  $r \equiv 0$ , i.e.  $b^{-\gamma r(s)} = 1$ . Here, every constant vector is a fixed point, however NegUtil\*  $\equiv 1$ . More generally, as the equations are purely multiplicative, for any fixed point  $v$ , every multiple  $\lambda \cdot v$  is a fixed point, too.

► **Example 7.** Again consider the example of Figure 2 with  $b = 2$  and  $\gamma = 1$ . The (simplified) equations we get are:

$$v_1 = 2^{-2} \cdot \min\{\frac{1}{2}v_2 + \frac{1}{2}v_3, v_4\} \quad v_2 = 2^{-2} \cdot v_2 \quad v_3 = v_3 \quad v_4 = 2^{-4} \cdot v_3,$$

where  $v_i$  corresponds to the value of  $s_i$ . First, for  $v_2$ , we observe that  $v_2 = 0$  is the only valid assignment. Then, we have that  $v_1 = 2^{-2} \cdot \min\{\frac{1}{2}0 + \frac{1}{2}v_3, 2^{-4}v_3\} = 2^{-3} \cdot \min\{v_3, 2^{-3}v_3\}$ . Clearly, this system is underdetermined and we obtain a distinct solution for any value of  $v_3$ .

To solve these issues, we need to define “anchors” of the equation. We observe the resemblance of classical fixed point equations for stochastic systems. In particular, for  $r \equiv 0$ , Equation (5) is the same as for reachability, Equation (1).

## 4.2 Reduction to Reachability

We define  $S_0 = \{s \mid \max_{\sigma} \min_{\tau} \Pr_{G,s}^{\sigma,\tau}[\text{TR} > 0] = 0\}$  and  $S_{\infty} = \{s \mid \max_{\sigma} \min_{\tau} \Pr_{G,s}^{\sigma,\tau}[\text{TR} = \infty] = 1\}$  the set of states in which Maximizer cannot obtain a total reward of more than 0 with positive probability against an optimal opponent strategy or ensure infinite reward with probability 1, respectively. We show later on that these sets are simple to compute and MD strategies are sufficient. Since  $r(s) \geq 0$ , all states in  $s \in S_0$  necessarily have  $r(s) = 0$ . Observe that  $S_0$  may be empty, but then  $S = S_{\infty}$  and so NegUtil\* = 0, ERisk\* =  $\infty$ . Through these sets, we can connect optimizing the utility to a reachability objective.

## 15:10 Entropic Risk for Turn-Based Stochastic Games

► **Lemma 8.** *For any state  $s$  in the game  $G$ , the optimal utility  $\text{NegUtil}^*$  is equal to the minimal probability of reaching the set  $S_0$  from  $s$  in game  $G_R$ , defined as follows: We add a designated sink state  $\underline{s}$  (which may belong to either player and only has a self-loop back to itself) and define  $\Delta_R(s, a, s') = b^{-\gamma r(s')} \cdot \Delta(s, a, s')$  for  $s, s' \in S$ ,  $a \in A(s)$  and  $\Delta_R(s, a, \underline{s}) = (1 - b^{-\gamma r(s)})$ . There is a direct correspondence between optimal strategies.*

We note that reachability games can also be reduced to our case:

► **Lemma 9.** *For any game  $G$  and (absorbing) reachability goal  $T$ , we have  $\text{Val}_{G, \diamond T}(s) = 1 - \text{NegUtil}_G^*(s)$  with reward  $r(s) = \mathbb{1}_T(s)$  and  $\gamma = 1$ .*

We highlight that this reduction from entropic risk games to reachability games is *not* an effective reduction in the computational sense, since  $G_R$  comprises *irrational* transition probabilities even for entirely rational inputs. We discuss how to tackle this in the next section and first proceed to derive some useful properties from this correspondence.

► **Lemma 10.** *The optimal utility  $\text{NegUtil}^*$  is the pointwise smallest solution of*

$$\begin{aligned} v(s) &= 0 \quad \text{for } s \in S_\infty, & v(s) &= 1 \quad \text{for } s \in S_0, \text{ and} \\ v(s) &= \overline{\text{opt}}_{a \in A(s)}^s b^{-\gamma r(s)} \cdot \Delta(s, a) \langle v \rangle & \text{otherwise} \end{aligned} \tag{6}$$

Yet, there might be multiple fixed points to the system of equations. This is to be expected, since already reachability on MDPs exhibits this problem [32]. We provide a discussion of these issues together with a sufficient condition for uniqueness in the extended version [3].

### 4.3 Strategy Complexity

By Lemma 8, the optimal negative exponential utility is achieved by reachability-optimal strategies in  $G_R$ . With the known results on reachability [18], this yields:

► **Theorem 11.** *MD strategies are sufficient to optimize the negative exponential utility and thus also entropic risk. More precisely, for all SGs  $G$ , there is an MD strategy  $\sigma$  for the Maximizer such that  $\text{ERisk}_{G,s}^{\gamma*} = \inf_{\tau \in \Pi_G} \text{ERisk}_{G,s}^\gamma((\sigma, \tau))$  and analogously for the Minimizer.*

► **Remark 12.** We highlight that this means that this notion of risk is history independent: Which actions are optimal does not depend on what has already “gone wrong”, but purely on the potential future consequences. This is in stark contrast to, e.g., conditional value-at-risk optimal strategies for total reward, which require exponential memory and switch to a purely expectation maximizing (i.e. risk-ignorant) behaviour after “enough” went wrong [42].

### 4.4 System of Inequalities

The problem we want to solve is deciding whether the Maximizer can ensure an entropic risk of at least  $t$ . Unfortunately, the reachability game  $G_R$  is not directly computable, since even for rational rewards  $b^{-\gamma r(s)}$  may be irrational. As such, we cannot use this transformation directly to prove decidability or complexity results and need to take a different route. Analogous to the classical solution to reachability, we first convert the problem to a system of inequalities. Intuitively, we replace every max with  $\geq$  for all options and dually min with  $\leq$  (again, recalling that Maximizer wants to minimize the value in  $G_R$ ). Formally, we consider the following:

$$\begin{aligned}
v(\hat{s}) &\leq b^{-\gamma t}, & v(s) = 0 & \text{ for } s \in S_\infty, & v(s) = 1 & \text{ for } s \in S_0, \\
v(s) &\leq b^{-\gamma r(s)} \cdot \Delta(s, a)\langle v \rangle & \text{ for } s \in S_{\max}, a \in A(s), \\
v(s) &\geq b^{-\gamma r(s)} \cdot \Delta(s, a)\langle v \rangle & \text{ for } s \in S_{\min}, a \in A(s), \text{ and} \\
\bigvee_{a \in A(s)} v(s) &= b^{-\gamma r(s)} \cdot \Delta(s, a)\langle v \rangle & \text{ for } s \in S
\end{aligned} \tag{7}$$

Observe that this essentially is the decision variant to the standard quadratic program for reachability applied to  $G_R$  [19].

► **Lemma 13.** *The system of equations 7 has a solution if and only if  $\text{ERisk}^* \geq t$ .*

## 4.5 Decidability Subject to Schanuel’s Conjecture

From Equation (7), we obtain a conditional decidability result for the general case:

► **Theorem 14.** *Let all quantities, i.e. rewards, transition probabilities, the risk-aversion factor  $\gamma$ , and the basis  $b$  be given as formulas in the language of reals with exponentiation (i.e. with functions  $+$ ,  $\cdot$ , and  $\exp: x \mapsto e^x$ ). Then, the entropic risk threshold problem for SGs is decidable subject to Schanuel’s conjecture.*

**Proof.** In this case, the existence of a solution to Equation (7) can also be expressed as a sentence in the language of the reals with exponentiation. The corresponding theory is known to be decidable subject to Schanuel’s conjecture (see e.g. [37]) as shown by [38], and decidability of this theory is equivalent to the so-called “weak Schanuel’s conjecture”. ◀

In particular, this allows us to treat instances with basis  $b = e$ . Yet, even if all rewards, transition probabilities, and  $\gamma$  are given as rational values, but the basis  $b$  equals  $e$ , we do not know how to check the satisfiability of Equation (7) without relying on the theory of the reals with exponentiation. Note, however, that we do not need the “full power” of the exponential function: All values appearing in an exponent in Equation (7) are constants. So, the restricted exponential function that agrees with  $\exp$  on a closed interval  $[a_1, a_2]$  and is zero outside of this interval is sufficient. The theory of the reals with restricted exponentiation has some additional nice properties compared to the theory of the reals with full exponentiation: For example, it allows for quantifier elimination by [52] and related works. Nevertheless, this does not allow us to immediately obtain an unconditional decidability result.

## 5 The Algebraic Case

If all occurring values are rational, then all numbers of the system of inequalities Equation (7) are algebraic. The results of this section establish that the threshold problem for such instances is decidable. A detailed exposition of the results can be found in the extended version [3]; an overview of the complexity results can also be found in Table 1. Formally, we define:

► **Definition 15.** *An algebraic instance of the entropic risk threshold problem is an instance where all occurring values, i.e. the transition probabilities of the game  $G$ , all rewards assigned by the reward function  $r$ , the risk-aversion parameter  $\gamma$ , the basis  $b$ , and the threshold  $t$ , are rational and encoded as the fraction of co-prime integers in binary.*

## 15:12 Entropic Risk for Turn-Based Stochastic Games

In general, for algebraic instances, there is a reduction of our problem to the existential theory of the reals, leading to the following result where  $\exists\mathbb{R}$  denotes the complexity class of problems that are polynomial-time reducible to the existential theory of the reals:

► **Theorem 16.** *For algebraic instances, the entropic risk threshold problem is decidable in  $\exists\mathbb{R}$  and thus in PSPACE.*

Already for Markov chains, it is unclear whether the upper complexity bound can be improved. For a discussion on this issue, see also the extended version [3].

For Theorem 16, we use the standard decision procedure for the existential theory of the reals as a “black box” and do not make use of the special form of our problem. To exploit the specific structure of the system of inequalities, we note that for explicit computations on algebraic numbers the following two quantities are relevant for the resulting computational complexity: Firstly, the degree of the field extension of  $\mathbb{Q}$  in which the computation can be carried out. Secondly, the bitsize of the coefficients of the minimal polynomials of the involved algebraic numbers (see, e.g., [1, 6]). Alternatively, the bitsize of the representations of the algebraic numbers in a fixed basis of the field extension in which the computations can be carried out can be used. Note that the size of the basis is precisely the degree of that field extension. Motivated by these observations, we consider *small algebraic instances*, which allow us to prove that all occurring algebraic numbers have a sufficiently small representation.

► **Definition 17.** *A small algebraic instance of the entropic risk threshold problem consists of a SG  $G$  with rational transition probabilities, an integer reward function  $r$ , a rational risk-aversion parameter  $\gamma$ , a rational basis  $b$ , and a rational threshold  $t$ . Moreover, the rewards,  $\gamma$ , and  $t$  are encoded in unary, and as the fraction of co-prime integers encoded in unary, respectively. The remaining rational numbers are encoded as the fraction of co-prime integers in binary. If  $G$  is an MDP or a MC, we call the instance a small algebraic instance of an MDP or a MC.*

► **Remark 18.** For simplicity, we assume for small algebraic instances that all rewards are in  $\mathbb{N}$ . If this is not the case, we can multiply all rewards with the least common multiple  $D$  of the denominators of the rewards and use a new risk-aversion parameter  $\gamma' = \gamma/D$ . The resulting negative exponential utility is not affected by this transformation. The change of the optimal entropic risk by a factor of  $D$  can be addressed by also rescaling the threshold  $t' = t \cdot D$ . Nevertheless, note that this affects the encoding size of the risk-aversion factor  $\gamma$ . Relying on algorithms for explicit computations in algebraic numbers [1, 6], we obtain:

► **Theorem 19.** *For small algebraic instances, the entropic risk threshold problem: (a) belongs to  $\text{NP} \cap \text{coNP}$  for SGs; and (b) can be solved in polynomial time for MDPs or MCs.*

While the mentioned results concern the threshold problem, we can even go a step further in small algebraic instances of MCs. Here, the system of inequalities simplifies to a linear system of equations, which we can solve *explicitly* in the algebraic numbers. For small algebraic instances, this is possible in polynomial time yielding the following result.

► **Theorem 20.** *For small algebraic instances, an explicit representation of  $\text{NegUtil}^*$  can be computed in: (a) polynomial time for MCs; and (b) in polynomial space for SGs and MDPs.*

## 6 Approximation Algorithms

The results of the previous section suggest, depending on the form of the input, a polynomial-space algorithm or even worse in the general case. Clearly, this is somewhat unsatisfactory for practical applications. Recall that the difficulties are due to the occurring irrational

transition probabilities. In the hope that we can work with approximations of these numbers, we now aim to identify an approach which allows us to approximate the correct answer, i.e. compute a value close to the optimal entropic risk that the Maximizer can ensure. Again, fix an SG  $G$ , reward function  $r$ , risk parameter  $\gamma$ , and risk basis  $b$  throughout this section. Then, given a precision  $\varepsilon > 0$ , we aim to compute a value  $v$  such that  $|\text{ERisk}^* - v| < \varepsilon$ , i.e. an approximation with small absolute error.

Since entropic risk is the logarithm of utility, we need to obtain an approximation of  $\text{NegUtil}^*$  to a sufficiently small *relative* error. Concretely, we need to compute a value  $v_U$  such that  $b^{-\gamma\varepsilon} \leq v_U / \text{NegUtil}^* \leq b^{\gamma\varepsilon}$ . Then,  $v = -\frac{1}{\gamma} \log_b(v_U)$  yields an approximation, since

$$\begin{aligned} \text{ERisk}^* - v &= -\frac{1}{\gamma} \log_b(\text{NegUtil}^*) + \frac{1}{\gamma} \log_b(v_U) = \frac{1}{\gamma} \log_b(v_U / \text{NegUtil}^*) = (*) \\ (*) &\geq \frac{1}{\gamma} \log_b(b^{-\gamma\varepsilon}) = -\varepsilon \quad \text{and} \quad (*) \leq \frac{1}{\gamma} \log_b(b^{\gamma\varepsilon}) = \varepsilon. \end{aligned}$$

(When we are interested in a concrete value for  $v$ , we need to determine  $v_U$  with a slightly higher precision and then approximate  $\log_b(v_U)$  sufficiently.) Now, in order to approximate  $\text{NegUtil}^*$ , we still need to deal with a system comprising potentially irrational transition probabilities. We argue that the occurring values  $b^{-\gamma r(s)}$  can be “rounded” to a sufficient precision while keeping the overall relative error small. Using techniques from [12], we provide an effective way to compute a game  $G_{\approx}$ , which behaves “similarly” to the reachability game  $G_R$  from Lemma 8, in the extended version [3]. Once  $G_{\approx}$  is computed, we can employ classical solution methods, such as linear equation solving for MCs, linear programming for MDPs, or, e.g., quadratic programming for SGs leading to an algorithm in polynomial space for SGs.

► **Theorem 21.** *In MCs and MDPs, the optimal value  $\text{ERisk}^*$  can be approximated up to an absolute error of  $\varepsilon$  in time polynomial in the size of the system,  $-\log(\varepsilon)$ ,  $\log b$ ,  $\gamma \cdot r_{\max}$ , and  $1/(\gamma \cdot r_{\min})$ , where  $r_{\max}$  and  $r_{\min}$  are the largest and smallest occurring non-zero rewards, respectively. For SGs, this is possible in polynomial space.*

In particular, for fixed  $b$  and  $\gamma$ , and bounded rewards (both from above and below), we obtain a PTIME solution for MC and MDP. In general, the procedure is exponential for SG. Alternatively, we can also apply different approaches such as value iteration [36].

► **Remark 22.** We note the connection to the small algebraic case: The “limiting factor” in both cases is the (size of the) product of  $\gamma$  and the state rewards. If these are fixed or given in unary, respectively, the complexity of our proposed algorithms is significantly reduced.

As a final note, recall that we do not assume  $\gamma$  or the transition probabilities to be rational. We only require that we can expand their binary representation to arbitrary precision. Then, we can conservatively approximate their logarithm to evaluate the required rounding precision and approximate the transition probabilities of  $G_{\approx}$  in the same way.

## 7 Conclusion

We applied the entropic risk to total rewards in SGs to capture risk-averse behavior in these games. The objective forces agents to achieve a good overall performance while keeping the chance of particularly bad outcomes small. We showed that SGs with the entropic risk as payoff function are determined and admit optimal MD-strategies. This reflects the time-consistency of entropic risk and makes entropic risk an appealing objective as, in contrast, the optimization of other risk-averse objective functions that have been studied on MDPs in the literature require strategies with large memory or complicated randomization.

Computationally, difficulties arise due to the involved exponentiation leading to irrational or even transcendental numbers. For the general case, we obtained decidability of the threshold problem only subject to Shanuel’s conjecture while for purely rational inputs, the problem can be solved via a reduction to the existential theory of the reals. Additional restrictions on the encoding of the input allowed us to obtain better upper bounds. Further, we provided an approximation algorithm for the optimal value. For an overview of the results, see Table 1.

A question that is left open is whether the entropic risk threshold problem for algebraic instances of MCs can be solved more efficiently than by the polynomial-time reduction to the existential theory of the reals. This case constitutes a bottleneck in the complexity. Furthermore, we worked with non-negative rewards, which made a reduction from games with the entropic risk objective to reachability games possible. Dropping the restriction to non-negative rewards constitutes an interesting direction of future research, in which additional difficulties arise and a reduction to reachability is not possible anymore. A further direction for future work is the experimental evaluation of the proposed algorithms to assess their practical applicability as well as to investigate the behavior of the resulting optimal strategies. In particular, it might be interesting to investigate the “cost” of risk-awareness, namely how much the expected total reward of a risk-aware strategy differs from a purely expectation maximizing one on realistic systems.

---

## References

- 1 Ilan Adler and Peter A. Beling. Polynomial algorithms for linear programming over the algebraic numbers. *Algorithmica*, 12(6):436–457, 1994.
- 2 Hubert Asienkiewicz and Anna Jaśkiewicz. A note on a new class of recursive utilities in Markov decision processes. *Applicationes Mathematicae*, 44:149–161, 2017.
- 3 Christel Baier, Krishnendu Chatterjee, Tobias Meggendorfer, and Jakob Piribauer. Entropic risk for turn-based stochastic games, 2023. arXiv preprint. [arXiv:2307.06611](https://arxiv.org/abs/2307.06611).
- 4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 5 Nicole Bäuerle and Ulrich Rieder. More risk-sensitive Markov decision processes. *Mathematics of Operations Research*, 39(1):105–120, 2014.
- 6 Peter A. Beling. Exact algorithms for linear programming over algebraic extensions. *Algorithmica*, 31(4):459–478, 2001.
- 7 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, 1991.
- 8 Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific, 1996. URL: <https://www.worldcat.org/oclc/35983505>.
- 9 Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.
- 10 Mario Brandtner, Wolfgang Kürsten, and Robert Rischau. Entropic risk measures and their comparative statics in portfolio selection: Coherence vs. convexity. *European Journal of Operational Research*, 264(2):707–716, 2018.
- 11 Tomáš Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017.
- 12 Krishnendu Chatterjee. Robustness of structurally equivalent concurrent parity games. In *International Conference on Foundations of Software Science and Computational Structures*, pages 270–285. Springer, 2012.
- 13 Krishnendu Chatterjee and Thomas A. Henzinger. Value iteration. In *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 107–138. Springer, 2008.
- 14 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic  $\omega$ -regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

- 15 Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.
- 16 Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods Syst. Des.*, 43(1):61–92, 2013. doi:10.1007/s10703-013-0183-7.
- 17 E.J. Collins. Finite-horizon variance penalised Markov decision processes. *Operations-Research-Spektrum*, 19(1):35–39, 1997.
- 18 Anne Condon. On algorithms for simple stochastic games. *Advances in computational complexity theory*, 13:51–72, 1990.
- 19 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.
- 20 Giovanni B Di Masi and Lukasz Stettner. Risk-sensitive control of discrete-time Markov processes with infinite horizon. *SIAM Journal on Control and Optimization*, 38(1):61–78, 1999.
- 21 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 2012.
- 22 Jerzy A. Filar, Lodewijk C. M. Kallenberg, and Huey-Miin Lee. Variance-penalized Markov decision processes. *Math. Oper. Res.*, 14(1):147–161, 1989.
- 23 Hans Föllmer and Thomas Knispel. Entropic risk measures: Coherence vs. convexity, model ambiguity and robust large deviations. *Stochastics and Dynamics*, 11(02n03):333–351, 2011.
- 24 Hans Föllmer and Alexander Schied. Convex measures of risk and trading constraints. *Finance and stochastics*, 6(4):429–447, 2002.
- 25 Hans Föllmer, Alexander Schied, and T Lyons. Stochastic finance. an introduction in discrete time. *The Mathematical Intelligencer*, 26(4):67–68, 2004.
- 26 Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011.
- 27 Liping Fu and Larry R Rilett. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological*, 32(7):499–516, 1998.
- 28 Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.
- 29 S. Gómez, A. Arenas, J. Borge-Holthoefer, S. Meloni, and Y. Moreno. Discrete-time Markov chain approach to contact-based disease spreading in complex networks. *EPL (Europhysics Letters)*, 89(3), February 2010.
- 30 Christoph Haase and Stefan Kiefer. The odds of staying on budget. In *ICALP (2)*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015.
- 31 Christoph Haase, Stefan Kiefer, and Markus Lohrey. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*, pages 1–12. IEEE Computer Society, 2017.
- 32 Serge Haddad and Benjamin Monmege. Interval iteration algorithm for mdps and imdps. *Theor. Comput. Sci.*, 735:111–131, 2018.
- 33 Ronald A Howard and James E Matheson. Risk-sensitive Markov decision processes. *Management science*, 18(7):356–369, 1972.
- 34 Stratton C Jaquette. A utility criterion for Markov decision processes. *Management Science*, 23(1):43–49, 1976.
- 35 Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *LICS*, pages 609–618. ACM, 2018.
- 36 Jan Kretínský, Tobias Meggendorfer, and Maximilian Weininger. Stopping criteria for value iteration on stochastic games with quantitative objectives. In *LICS*, 2023.
- 37 Serge Lang. *Introduction to transcendental numbers*. Addison-Wesley Publishing Company, 1966.

- 38 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.
- 39 A Maitra and W Sudderth. Stochastic games with borel payoffs. In *Stochastic Games and Applications*, pages 367–373. Springer, 2003.
- 40 Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In *ICML*, pages 177–184. Omnipress, 2011.
- 41 Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.
- 42 Tobias Meggendorfer. Risk-aware stochastic shortest path. In *AAAI*, pages 9858–9867. AAAI Press, 2022.
- 43 Johan Paulsson. Summing up the noise in gene networks. *Nature*, 427(6973):415–418, 2004.
- 44 Jakob Piribauer and Christel Baier. On skolem-hardness and saturation points in Markov decision processes. In *ICALP*, volume 168 of *LIPICs*, pages 138:1–138:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 45 Jakob Piribauer, Ocan Sankur, and Christel Baier. The variance-penalized stochastic shortest path problem. In *ICALP*, volume 229 of *LIPICs*, pages 129:1–129:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 46 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.
- 47 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Variations on the stochastic shortest path problem. In *VMCAI*, volume 8931 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2015.
- 48 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.
- 49 Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.
- 50 Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. Incremental plan aggregation for generating policies in mdps. In *AAMAS*, pages 1231–1238. IFAAMAS, 2010.
- 51 Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013.
- 52 Lou van den Dries, Angus Macintyre, and David Marker. The elementary theory of restricted analytic fields with exponentiation. *Annals of Mathematics*, 140(1):183–205, 1994.
- 53 Maximilian Weininger, Tobias Meggendorfer, and Jan Kretínský. Satisfiability bounds for  $\omega$ -regular properties in bounded-parameter Markov decision processes. In *CDC*, pages 2284–2291. IEEE, 2019.

# Speed Me up If You Can: Conditional Lower Bounds on Opacity Verification

Jiří Balun ✉ 

Faculty of Science, Palacky University Olomouc, Czech Republic

Tomáš Masopust ✉ 

Faculty of Science, Palacky University Olomouc, Czech Republic

Petr Osička ✉ 

Faculty of Science, Palacky University Olomouc, Czech Republic

---

## Abstract

Opacity is a property of privacy and security applications asking whether, given a system model, a passive intruder that makes online observations of system's behaviour can ascertain some “secret” information of the system. Deciding opacity is a PSPACE-complete problem, and hence there are no polynomial-time algorithms to verify opacity under the assumption that PSPACE differs from PTIME. This assumption, however, gives rise to a question whether the existing exponential-time algorithms are the best possible or whether there are faster, sub-exponential-time algorithms. We show that under the (Strong) Exponential Time Hypothesis, there are no algorithms that would be significantly faster than the existing algorithms. As a by-product, we obtained a new conditional lower bound on the time complexity of deciding universality (and therefore also inclusion and equivalence) for nondeterministic finite automata.

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

**Keywords and phrases** Finite automata, opacity, fine-grained complexity

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.16

**Related Version** *Full Version*: <https://doi.org/10.48550/arXiv.2304.09920> [7]

**Funding** *Jiří Balun*: Supported by IGA PrF 2023 026.

**Acknowledgements** We acknowledge valuable comments and suggestions of anonymous referees.

## 1 Introduction

In privacy and security applications and communication protocols, it is desirable to keep some information about the system or its behaviour secret. Such requirements put additional restrictions on the information flow of the system and have widely been discussed in the literature as various properties, including *anonymity* [14, 32, 38], *non-interference* [8, 12, 19, 33], *secrecy* [1, 3, 10], *security* [17], *perfect security* [46], and *opacity* [20, 30]. *Anonymity* is the property to preserve secrecy of identity of actions; for instance, web servers should not be able to learn the true source of a request. *Non-interference* asks whether, given two input states of the system that share the same values of specified variables, the behaviors of the system started from these states are indistinguishable by the observer under the observation of the specified variables. *Secrecy* expresses whether an observer can ever find out that a trajectory of the system belongs to a set of secret trajectories, and *perfect security* requires that an observer that knows the set of all trajectories of the system cannot deduce any information about occurrences of high-security events by observing low-security events.

In this paper, we are interested in (various types of) opacity, which in a sense generalizes the other mentioned properties; namely, the properties above can be verified by reduction to opacity. More specifically, Alur et al. [1] have shown that *secrecy* captures *non-interference*



© Jiří Balun, Tomáš Masopust, and Petr Osička;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 16; pp. 16:1–16:15

Leibniz International Proceedings in Informatics



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

and *perfect security*, and Lin [26] has provided an extensive discussion and comparison of all these properties. He has in particular shown that *anonymity* and *secrecy*, and the properties of *observability* [27], *diagnosability* [25, 37], and *detectability* [28, 29, 39] of discrete-event systems are special cases of opacity. Wu et al. [44] and Góes et al. [18] discuss applications of opacity in location privacy, and Wintenberg et al. [42] apply opacity in contact tracing.

Given a system model, *opacity* asks whether a malicious passive observer (an intruder) with a complete knowledge of the structure of the model can ascertain some “secret” information of the system by making online incomplete observations of system’s behaviour. The secret information is modeled either as a set of states or as a set of behaviours of the system. Based on the incomplete observations, the intruder estimates the state/behaviour of the system, and the system is opaque if for every secret state/behaviour of the system, there is a non-secret state/behaviour of the system that looks the same to the intruder. If the secret information is given as a set of states, we talk about state-based opacity [9, 11], whereas if the secret information is given as a set of secret behaviours (a language), we talk about language-based opacity [3, 15]. Several notions of opacity have been discussed in the literature for systems modeled by automata and Petri nets; see Jacob et al. [24] for an overview. In this paper, we focus on finite automata models and on the notions of opacity that we review in Section 3.

The fastest existing algorithms verifying the notions of opacity under consideration have exponential-time complexity with respect to the number of states of the automaton. In fact, the verification of opacity is a PSPACE-complete problem [4, 5, 24], and hence we may conclude that there are no polynomial-time algorithms deciding opacity unless PTIME = PSPACE. Although the assumption that PTIME  $\neq$  PSPACE excludes the existence of polynomial-time algorithms, the question whether there is a significantly faster (i. e., sub-exponential-time) algorithm remains open.

To achieve stronger lower bounds (although still conditional), we use the *Exponential Time Hypothesis* (ETH) and its strong version – the *Strong Exponential Time Hypothesis* (SETH). Both hypotheses were formulated by Impagliazzo and Paturi [23] and are based on the observation that (so far) we were not able to find algorithms that would, in the worst case, solve SAT significantly faster than the algorithms trying all possible truth assignments. In particular, ETH states that 3-SAT cannot be solved in time  $2^{o(n)}$  where  $n$  is the number of variables. However, it admits algorithms solving 3-SAT in time  $O(c^n)$  where  $c < 2$ . In fact, the current fastest 3-SAT algorithm of Paturi et al. [31], improved by Hertli [21], runs in time  $O^*(1.30704^n)$ . With increasing  $k$ , the current fastest  $k$ -SAT algorithms are getting slower; for instance, the best 4-SAT algorithm of Hertli [21] runs in time  $O^*(1.46899^n)$ . This observation motivated the formulation of SETH that claims that, for any constant  $c < 2$ , there is always a sufficiently large  $k$  such that  $k$ -SAT cannot be solved in time  $O(c^n)$  [23]. Both hypotheses imply that the complexity classes PTIME and NP are separated; moreover, SETH implies ETH.

In this paper, we show that under (S)ETH, there are no significantly faster algorithms verifying opacity. In particular, we show that unless SETH fails, there is no algorithm that decides whether a given  $n$ -state automaton satisfies the considered notions of opacity and runs in time  $O^*(2^{n/c})$ , for any  $c > 2$  (Theorem 8 and Corollary 15). Since the number of symbols in the alphabet of our construction is unbounded and the standard binary encoding of symbols does not work under SETH, it is not clear whether this result also holds for automata with a fixed size (binary) alphabet. We partially explore this question under ETH rather than SETH. We show that unless ETH fails, there is no algorithm that decides whether a given  $n$ -state automaton (over a binary alphabet) satisfies the considered notions of opacity and runs in time  $O^*(2^{o(n)})$ . Our results are summarized in Table 1; for the complexity upper bounds, we refer the reader to the literature [5, 34].

■ **Table 1** An overview of the algorithmic complexity of deciding opacity under the projection  $P: \Sigma^* \rightarrow \Gamma^*$ , where  $n$  is the number of states of the automaton and  $\varepsilon > 0$ .

	Lower bound		Upper bound
	$\Gamma$ not fixed	$ \Gamma  = 2$	
LBO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$O^*(2^n)$
CSO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$O^*(2^n)$
ISO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$O^*(2^n)$
IFO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$\begin{cases} O^*(2^n) & \text{if } IQ_{NS} = I_{NS} \times F_{NS} \\ O^*(2^{n^2}) & \text{otherwise} \end{cases}$
$k$ -SO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$O^*(2^n)$
INSO	$O^*(2^{n/(2+\varepsilon)})$	$O^*(2^{o(n)})$	$O^*(2^n)$

As a by-product, we obtain a new conditional lower bound for deciding universality (and hence inclusion and equivalence) for nondeterministic automata (NFA): Unless SETH fails, there is no  $c > 2$  such that the universality of an  $n$ -state NFA can be decided in time  $O^*(2^{n/c})$  (Corollary 18). This result strengthens the result of Fernau and Krebs [16] showing that if ETH is true, the universality of an  $n$ -state NFA cannot be decided in time  $O^*(2^{o(n)})$ .

## 2 Preliminaries

We assume that the reader is familiar with automata theory [22]. For a set  $S$ , the cardinality of  $S$  is denoted by  $|S|$  and the power set of  $S$  by  $2^S$ . If  $S$  is a singleton,  $S = \{x\}$ , we often simply write  $x$  instead of  $\{x\}$ . The set of all non-negative integers is denoted by  $\mathbb{N}$ .

An alphabet  $\Sigma$  is a finite nonempty set of symbols. A string over  $\Sigma$  is a finite sequence of symbols from  $\Sigma$ . The set of all strings over  $\Sigma$  is denoted by  $\Sigma^*$ ; the empty string is denoted by  $\varepsilon$ . A language  $L$  over  $\Sigma$  is a subset of  $\Sigma^*$ . For a string  $u \in \Sigma^*$ , the length of  $u$  is denoted by  $|u|$ . With every pair of alphabets  $(\Sigma, \Gamma)$  with  $\Gamma \subseteq \Sigma$ , we associate the morphism  $P: \Sigma^* \rightarrow \Gamma^*$  defined by  $P(a) = \varepsilon$ , for  $a \in \Sigma - \Gamma$ , and  $P(a) = a$ , for  $a \in \Gamma$ ; such morphisms are usually called *projections*. Intuitively, the action of the projection  $P$  is to erase all symbols that do not belong to  $\Gamma$ ; the symbols of  $\Gamma$  are usually called *observable symbols of  $\Sigma$  under the projection  $P$* . We lift the projection  $P$  from strings to languages in the usual way. The inverse projection of  $P$  is the function  $P^{-1}: \Gamma^* \rightarrow 2^{\Sigma^*}$  defined by  $P^{-1}(w) = \{w' \in \Sigma^* \mid P(w') = w\}$ .

A *nondeterministic finite automaton* (NFA) is a structure  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is an input alphabet,  $I \subseteq Q$  is a set of initial states,  $F \subseteq Q$  is a set of accepting states, and  $\delta: Q \times \Sigma \rightarrow 2^Q$  is a transition function that can be extended to the domain  $2^Q \times \Sigma^*$  by induction. If the accepting states are irrelevant, we omit them and simply write  $\mathcal{A} = (Q, \Sigma, \delta, I)$ . The language accepted by  $\mathcal{A}$  from the states of  $Q_0 \subseteq Q$  by the states of  $F_0 \subseteq F$  is the set  $L_m(\mathcal{A}, Q_0, F_0) = \{w \in \Sigma^* \mid \delta(Q_0, w) \cap F_0 \neq \emptyset\}$  and the language generated by  $\mathcal{A}$  from the states of  $Q_0$  is the set  $L(\mathcal{A}, Q_0) = L_m(\mathcal{A}, Q_0, Q)$ ; in particular, the language accepted by  $\mathcal{A}$  is  $L_m(\mathcal{A}) = L_m(\mathcal{A}, I, F)$  and the language generated by  $\mathcal{A}$  is  $L(\mathcal{A}) = L(\mathcal{A}, I)$ . The NFA  $\mathcal{A}$  is *deterministic* (DFA) if  $|I| = 1$  and  $|\delta(q, a)| \leq 1$  for every state  $q \in Q$  and every symbol  $a \in \Sigma$ .

For an alphabet  $\Gamma \subseteq \Sigma$ , we define the *projected automaton* of  $\mathcal{A}$ , denoted by  $P(\mathcal{A})$ , as the reachable part of a DFA obtained from  $\mathcal{A}$  by replacing every transition  $(q, a, r)$  with the transition  $(q, P(a), r)$ , followed by the standard subset construction [22]. The projected automaton of  $\mathcal{A}$  is also known as the *observer* of  $\mathcal{A}$  in the literature.

We define the *configuration of  $\mathcal{A}$*  as the state of the projected automaton  $P(\mathcal{A})$  of  $\mathcal{A}$ .

A (*Boolean*) *formula* consists of variables, symbols for logical connectives: conjunction, disjunction, negation; and parentheses. A *literal* is a variable or its negation. A *clause* is a disjunction of literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. If each clause has at most  $k$  literals, the formula is in  $k$ -CNF. A formula is *satisfiable* if there is an assignment of 1 and 0 to the variables evaluating the formula to 1. Given a  $k \geq 3$  and a formula in  $k$ -CNF, the  $k$ -CNF *Boolean satisfiability problem* ( $k$ -SAT) is to decide whether the formula is satisfiable. If the formula in  $k$ -CNF has  $n$  variables, enumerating all the  $2^n$  possible truth assignments results in an  $O(2^n n^k)$ -time algorithm for  $k$ -SAT; the polynomial part  $O(n^k)$  comes from checking up to  $n^k$  clauses. We use the notation  $O^*$  to hide polynomial factors, that is,  $O^*(g(n)) = O(g(n) \cdot \text{poly}(n))$ .

The exponential time hypothesis states that 3-SAT cannot be solved in sub-exponential time  $2^{o(n)}$ , where  $n$  is the number of variables in the 3-CNF formula [23].

► **Hypothesis 1** (Exponential Time Hypothesis (ETH)). *There is some  $\varepsilon > 0$  such that 3-SAT cannot be solved in time  $O(2^{\varepsilon n})$ , where  $n$  is the number of variables in the formula.*

The strong ETH states that deciding  $k$ -SAT needs  $O^*(2^n)$  time for large  $k$  [23].

► **Hypothesis 2** (Strong ETH (SETH)). *For every  $\varepsilon > 0$ , there is some  $k \geq 3$  such that  $k$ -SAT cannot be solved in time  $O(2^{(1-\varepsilon)n})$ .*

### 3 Opacity Definitions

We now review the notions of opacity considered in this paper. We distinguish two types of opacity: those representing the secret by strings and those representing the secret by states.

Language-based opacity is a property asking whether for every secret behaviour, there is a non-secret behaviour that is the same under a considered projection; in this case, an intruder cannot distinguish the secret behaviour from a non-secret behaviour.

► **Definition 3.** *An NFA  $\mathcal{A} = (Q, \Sigma, \delta, I)$  is language-based opaque (LBO) with respect to disjoint languages  $L_S, L_{NS} \subseteq L(\mathcal{A})$ , called secret and non-secret languages, respectively, and a projection  $P: \Sigma^* \rightarrow \Gamma^*$  for  $\Gamma \subseteq \Sigma$ , if  $L_S \subseteq P^{-1}P(L_{NS})$ .*

*The LBO problem is to decide whether  $\mathcal{A}$  is LBO with respect to  $L_S, L_{NS}$ , and  $P$ .*

Although this definition is a special case of the most general notion of *trace opacity* of Bryans et al. [9], it is general enough to capture other notions, such as *strong nondeterministic non-interference* or *non-deducibility on composition* of Best et al. [8] and Busi and Gorrieri [12]. The secret and non-secret languages are often considered to be regular to ensure that the inclusion problem is decidable [2].

State-based opacity hides the secret information into states. In this paper, we consider five notions of state-based opacity. Current-state opacity requires that an intruder cannot identify, at any instance of time, whether the system is currently in a secret state.

► **Definition 4.** *An NFA  $\mathcal{A} = (Q, \Sigma, \delta, I)$  is current-state opaque (CSO) with respect to two disjoint sets  $Q_S, Q_{NS} \subseteq Q$  of secret and non-secret states, respectively, and a projection  $P: \Sigma^* \rightarrow \Gamma^*$  for  $\Gamma \subseteq \Sigma$ , if for every string  $w \in \Sigma^*$  such that  $\delta(I, w) \cap Q_S \neq \emptyset$ , there exists a string  $w' \in \Sigma^*$  such that  $P(w) = P(w')$  and  $\delta(I, w') \cap Q_{NS} \neq \emptyset$ .*

*The CSO problem is to decide whether  $\mathcal{A}$  is CSO with respect to  $Q_S, Q_{NS}$ , and  $P$ .*

Initial-state opacity requires that an intruder can never ascertain whether the computation started in a secret state.

► **Definition 5.** An NFA  $\mathcal{A} = (Q, \Sigma, \delta, I)$  is initial-state opaque (ISO) with respect to two disjoint sets  $I_S, I_{NS} \subseteq I$  of secret and non-secret initial states, respectively, and a projection  $P: \Sigma^* \rightarrow \Gamma^*$  for  $\Gamma \subseteq \Sigma$ , if for every  $w \in L(\mathcal{A}, I_S)$ , there exists  $w' \in L(\mathcal{A}, I_{NS})$  such that  $P(w) = P(w')$ .

The ISO problem is to decide whether  $\mathcal{A}$  is ISO with respect to  $I_S, I_{NS}$ , and  $P$ .

Initial-and-final-state opacity [43] generalizes both CSO and ISO. The secret is represented as a set of pairs of an initial state and of an accepting state. Therefore, ISO is a special case of initial-and-final-state opacity where the accepting states do not play a role, while CSO is a special case where the initial states do not play a role.

► **Definition 6.** An NFA  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  is initial-and-final-state opaque (IFO) with respect to two disjoint sets  $IQ_S, IQ_{NS} \subseteq I \times F$  of secret and non-secret pairs of states, respectively, and a projection  $P: \Sigma^* \rightarrow \Gamma^*$  for  $\Gamma \subseteq \Sigma$ , if for every secret pair  $(q_0, q_f) \in IQ_S$  and every string  $w \in L_m(\mathcal{A}, q_0, q_f)$ , there exists a non-secret pair  $(q'_0, q'_f) \in IQ_{NS}$  and a string  $w' \in L_m(\mathcal{A}, q'_0, q'_f)$  such that  $P(w) = P(w')$ .

The IFO problem is to decide whether  $\mathcal{A}$  is IFO with respect to  $IQ_S, IQ_{NS}$ , and  $P$ .

The algorithmic time complexity of deciding IFO is known to be  $O^*(2^{n^2})$  in general, and  $O(2^{2n})$  if  $IQ_S = I_S \times F_S$  and  $IQ_{NS} = I_{NS} \times F_{NS}$ , for some  $I_S, I_{NS} \subseteq I$  and  $F_S, F_{NS} \subseteq F$  [43]. Our complexity in Table 1 is based on the following observations.

Consider an NFA  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  and two sets  $IQ_S, IQ_{NS} \subseteq I \times F$ . The IFO property of  $\mathcal{A}$  is unchanged if all pairs  $(s, f_1), (s, f_2), \dots, (s, f_k)$  with a common left component are replaced by a single pair of the form  $(s, \{f_1, f_2, \dots, f_k\})$ . This reduces the number of pairs to be considered to  $n$ , where  $n$  is the number of states of  $\mathcal{A}$ . For every pair  $(s_i, F_i) \in I \times 2^F$ , we define the language  $L_i = L_m(\mathcal{A}, s_i, F_i)$  and the languages

$$L_S = \bigcup_{(s_i, F_i) \in IQ_S} L_i \quad \text{and} \quad L_{NS} = \bigcup_{(s_i, F_i) \in IQ_{NS}} L_i.$$

Then, deciding whether  $\mathcal{A}$  is IFO with respect to  $IQ_S, IQ_{NS}$ , and  $P$  is equivalent to deciding whether the inclusion  $P(L_S) \subseteq P(L_{NS})$  holds true. Since both  $L_S$  and  $L_{NS}$  can be represented by NFAs consisting of at most  $n$  copies of  $\mathcal{A}$ , they have  $O(n^2)$  states. The inclusion  $P(L_S) \subseteq P(L_{NS})$  of languages of two NFAs can be tested in time  $O(n^2 2^{n^2}) = O^*(2^{n^2})$ , which is a complexity upper bound that coincides with the bound of Saboori and Hadjicostis [36], who used trellis automata.

If  $IQ_{NS} = I_{NS} \times F_{NS} \subseteq I \times F$ , then the NFA for  $L_{NS}$  coincides with  $\mathcal{A}$  where the initial states are  $I_{NS}$  and the final states are  $F_{NS}$ . In particular, this automaton has  $n$  states, and therefore the inclusion  $P(L_S) \subseteq P(L_{NS})$  can be tested in time  $O(n^2 2^n) = O^*(2^n)$ .

The notion of  $k$ -step opacity generalizes CSO by requiring that the intruder cannot ascertain the secret in the current and  $k$  subsequent states. By definition, CSO is equivalent to 0-step opacity. We use a slight generalisation of a definition of Saboori and Hadjicostis [35] that was formulated by Balun and Masopust [5].

► **Definition 7.** An NFA  $\mathcal{A} = (Q, \Sigma, \delta, I)$  is  $k$ -step opaque ( $k$ -SO), for a given  $k \in \mathbb{N} \cup \{\infty\}$ , with respect to two disjoint sets  $Q_S, Q_{NS} \subseteq Q$  of secret and non-secret states, respectively, and a projection  $P: \Sigma^* \rightarrow \Gamma^*$  for  $\Gamma \subseteq \Sigma$ , if for every string  $st \in L(\mathcal{A})$  such that  $|P(t)| \leq k$  and  $\delta(\delta(I, s) \cap Q_S, t) \neq \emptyset$ , there exists a string  $s't' \in L(\mathcal{A})$  such that  $P(s) = P(s')$ ,  $P(t) = P(t')$ , and  $\delta(\delta(I, s') \cap Q_{NS}, t') \neq \emptyset$ .

The  $k$ -SO problem is to decide whether  $\mathcal{A}$  is  $k$ -SO with respect to  $Q_S, Q_{NS}$ , and  $P$ .

A special case of  $k$ -SO for  $k$  being infinity is called infinite-step opacity (INSO). These two notions are closely related for finite automata, because an  $n$ -state automaton is infinite-step opaque if and only if it is  $(2^n - 2)$ -step opaque [45].

For the state-based opacity notions, we may assume, without loss of generality, that the projection  $P$  is an identity; indeed, the transitions labeled by symbols from  $\Sigma - \Gamma$  can be seen as  $\varepsilon$ -transitions, and hence they can be removed by the classical algorithm eliminating  $\varepsilon$ -transitions [22]. This algorithm does not change the number of states but can quadratically increase the number of transitions. In the sequel, we omit the projection if it is an identity.

#### 4 Lower Bounds under Strong Exponential Time Hypothesis

We now show that under the strong exponential time hypothesis, there is no algorithm deciding current-state opacity that would be significantly faster than the best known algorithm.

► **Theorem 8.** *Unless SETH fails, there is no algorithm deciding whether a given  $n$ -state NFA is CSO that runs in time  $O^*(2^{n/(2+\varepsilon)})$ , for any  $\varepsilon > 0$ .*

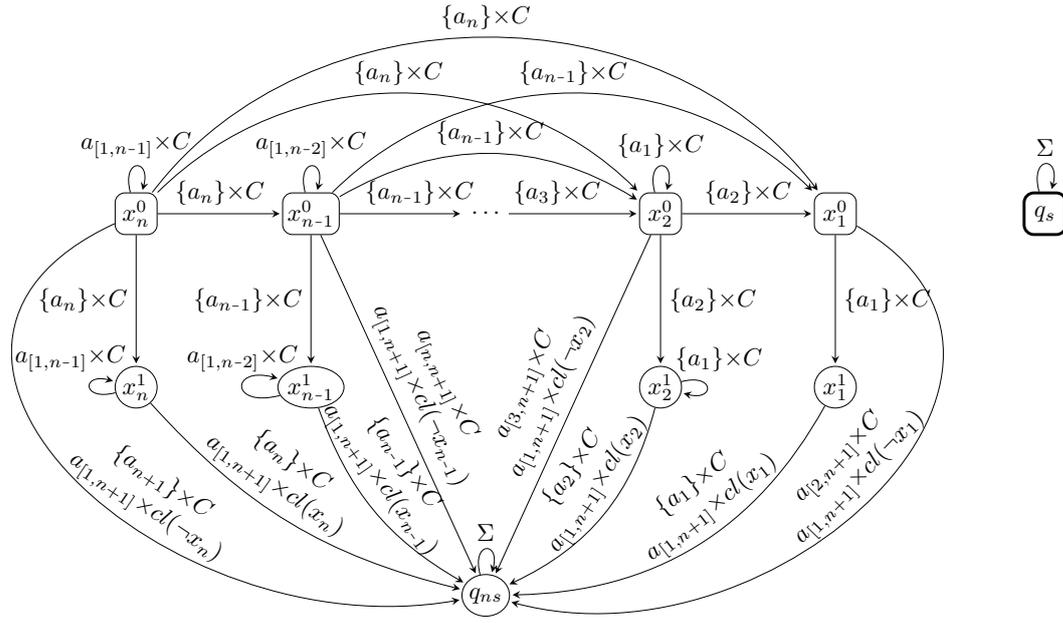
**Proof.** For a given formula  $\varphi$  in  $k$ -CNF with  $n$  variables  $X = \{x_1, \dots, x_n\}$  and  $m$  clauses  $C = \{c_1, \dots, c_m\}$ , we construct, in polynomial time, an instance of CSO consisting of an NFA  $\mathcal{A}_\varphi$  with  $N = 2n + 2$  states and of sets of secret and non-secret states  $Q_S$  and  $Q_{NS}$ , respectively, such that  $\mathcal{A}_\varphi$  is CSO with respect to  $Q_S$  and  $Q_{NS}$  if and only if  $\varphi$  is satisfiable. As a result, if there was an algorithm solving CSO in time  $O^*(2^{N/(2+\varepsilon)})$ , then there would be an algorithm solving  $k$ -SAT in time  $O(\text{poly}(n)) + O^*(2^{(2n+2)/(2+\varepsilon)}) = O^*(2^{(1-\delta)n})$ , for  $\delta = \varepsilon/(2+\varepsilon) > 0$ , which contradicts SETH, and proves the theorem.

Intuitively, we construct the NFA  $\mathcal{A}_\varphi$  such that when  $\mathcal{A}_\varphi$  reads a string of a particular type (based on Zimin words), it is forced to examine all possible assignments to the variables  $x_1, \dots, x_n$ . If none of the assignments satisfies  $\varphi$ , then, after reading the whole string, the automaton  $\mathcal{A}_\varphi$  ends up in a configuration that contains only secret states, rendering thus  $\mathcal{A}_\varphi$  not CSO. On the other hand, when a satisfying assignment is encountered (or the string is not of the particular type), a non-secret state is permanently added to the configuration of  $\mathcal{A}_\varphi$ , and hence  $\mathcal{A}_\varphi$  is CSO.

Formally, the NFA  $\mathcal{A}_\varphi = (Q, \Sigma, \delta, I)$ , where the set of states is  $Q = \{q_s, q_{ns}\} \cup \{x_i^0, x_i^1 \mid x_i \in X\}$  with  $x_i^r$  representing the assignment of  $r \in \{0, 1\}$  to the variable  $x_i$ , the alphabet  $\Sigma = \Gamma = \{a_1, \dots, a_{n+1}\} \times C$ , that is, the projection  $P$  is an identity, and the set of secret states is  $Q_S = \{q_s\}$ , that is, the state  $q_s$  is the only secret state, the remaining states are non-secret. For an illustration of the construction, the reader may follow Example 14 together with the rest of the proof.

Let  $L$  be the set of literals of  $\varphi$ . We use the function  $cl: L \rightarrow 2^C$  that assigns to a literal  $\ell$  the set  $cl(\ell) = \{c \in C \mid \ell \in c\}$  of clauses containing  $\ell$ , and define the transition function  $\delta$  as follows, see Figure 1 for an illustration:

- The self-loops  $(q_s, \sigma, q_s)$  and  $(q_{ns}, \sigma, q_{ns})$  belong to  $\delta$  for every  $\sigma \in \Sigma$ ;
- For every state  $x_i^0$  and every  $c \in C$ ,
  - the transition  $(x_i^0, (a_i, c), x_i^1) \in \delta$ ;
  - the self-loop  $(x_i^0, (a_j, c), x_i^0) \in \delta$  for  $1 \leq j \leq i - 1$ ;
  - the transition  $(x_i^0, (a_i, c), x_j^0) \in \delta$  for  $1 \leq j \leq i - 1$ ;
  - the transition  $(x_i^0, (a_j, c), q_{ns}) \in \delta$  for  $i + 1 \leq j \leq n + 1$ ;
  - the transition  $(x_i^0, (a_j, c), q_{ns}) \in \delta$  for  $1 \leq j \leq n + 1$  and  $c \in cl(\neg x_i)$ ;
- For every state  $x_i^1$  and  $c \in C$ ,
  - the transition  $(x_i^1, (a_i, c), q_{ns}) \in \delta$ ;
  - the self-loop  $(x_i^1, (a_j, c), x_i^1) \in \delta$ , for  $1 \leq j \leq i - 1$ ;
  - the transition  $(x_i^1, (a_j, c), q_{ns}) \in \delta$  for  $1 \leq j \leq n + 1$  and  $c \in cl(x_i)$ .



■ **Figure 1** The NFA  $A_\varphi$  of Theorem 8, where the initial states are squared, the single secret state  $q_s$  is in bold, and for positive integers  $i \leq j$ ,  $[i, j] = \{i, i + 1, \dots, j\}$  and  $a_{[i,j]} = \{a_r \mid r \in [i, j]\}$ .

Finally, the set of initial states is  $I = \{q_s\} \cup \{x_1^0, \dots, x_n^0\}$ , corresponding to the assignment of 0 to all variables of  $\varphi$ .

We now define a language  $W_\varphi = Z_n \cdot (\{a_{n+1}\} \times C)$ , where  $Z_n$  is a language over the alphabet  $\{a_1, a_2, \dots, a_n\} \times C$  recursively defined as follows:

$$Z_1 = \{a_1\} \times C \quad \text{and} \quad Z_i = Z_{i-1} \cdot (\{a_i\} \times C) \cdot Z_{i-1}, \quad \text{for } 1 < i \leq n.$$

Such strings are known as *Zimin words* and it is well known that any string of  $Z_n$  is of length  $2^n - 1$  [40] and that

$$\text{the symbol on the } \ell\text{th position of any string from } Z_n \text{ is of the form } \{a_j\} \times C, \text{ where } j - 1 \text{ is the number of trailing zeros in the binary representation of } \ell \text{ [41].} \quad (1)$$

We finish the proof in a series of claims. The first claim shows that along any string of  $Z_n$ , the states  $\{x_i^0, x_i^1 \mid x_i \in X\}$  of  $A_\varphi$  encode all possible assignments to the variables.

▷ **Claim 9.** Let  $A_\varphi^X$  denote  $A_\varphi$  without the states  $q_s$  and  $q_{ns}$  and the corresponding transitions. For every  $w \in Z_n$ , after reading the prefix of  $w$  of length  $\ell \leq 2^n - 1$ , the configuration of  $A_\varphi^X$  is  $\{x_n^{r_n}, x_{n-1}^{r_{n-1}}, \dots, x_1^{r_1}\}$ , where  $r_n r_{n-1} \dots r_1$  represents  $\ell$  in binary.

▷ **Claim 10.** Every configuration of  $A_\varphi$  contains the secret state  $q_s$ .

By Claims 9 and 10, and because only the state  $q_s$  itself is reachable from  $q_s$  and only the state  $q_{ns}$  itself is reachable from  $q_{ns}$ , we have the following observation specifying the computation of  $A_\varphi$  along the strings of  $Z_n$ .

▷ **Claim 11.** After reading the prefix of  $w \in Z_n$  of length  $\ell \leq 2^n - 1$ , the configuration of  $A_\varphi$  is either  $\{x_n^{r_n}, x_{n-1}^{r_{n-1}}, \dots, x_1^{r_1}\} \cup \{q_s\}$  or  $\{x_n^{r_n}, x_{n-1}^{r_{n-1}}, \dots, x_1^{r_1}\} \cup \{q_s, q_{ns}\}$ , where  $r_n r_{n-1} \dots r_1$  represents  $\ell$  in binary. ◁

If there is a satisfying assignment, then the non-secret state  $q_{ns}$  is reached by  $\mathcal{A}_\varphi$ .

▷ **Claim 12.** For every prefix  $w$  of a string in  $Z_n$ , if the configuration of  $\mathcal{A}_\varphi$  after reading  $w$  satisfies  $\varphi$ , then after reading any further symbol in  $\Sigma$ , the configuration of  $\mathcal{A}_\varphi$  contains  $q_{ns}$ .

We now show that if  $\varphi$  is satisfiable, then  $\mathcal{A}_\varphi$  is CSO. To this end, we consider an arbitrary string  $w$  over  $\Sigma$ , and we denote by  $u$  the longest prefix of  $w$  that is a prefix of a string in  $Z_n$ . Let  $\ell$  be the minimal number such that its binary representation  $r_n r_{n-1} \cdots r_1$  is a satisfying assignment to the variables of  $\varphi$ .

If  $\ell \leq |u|$ , then, by Claim 11, the configuration of  $\mathcal{A}_\varphi$  after reading any prefix of  $u$  of length  $\ell' \leq \ell$  contains non-secret states  $x_n^{r_n}, x_{n-1}^{r_{n-1}}, \dots, x_1^{r_1}$ , where  $r_n r_{n-1} \cdots r_1$  represents  $\ell'$  in binary, and, by Claim 12, the  $(\ell + 1)$ st symbol of  $w$  moves  $\mathcal{A}_\varphi$  to a configuration that contains  $q_{ns}$ ; that is,  $\mathcal{A}_\varphi$  is CSO.

If  $\ell > |u|$ , we have  $w = u(a_s, c)v$  for  $(a_s, c) \in \Sigma$  and  $v \in \Sigma^*$ . Because  $\varphi$  is satisfiable, we have  $|u| < 2^n - 1$ . By Claim 11, the configuration of  $\mathcal{A}_\varphi$  after reading  $u$  contains  $x_n^{r_n}, \dots, x_1^{r_1}$ , where  $r_n r_{n-1} \cdots r_1$  represents  $|u|$  in binary. Let  $r_t$  be the rightmost zero of  $r_n r_{n-1} \cdots r_1$ , that is,  $r_n r_{n-1} \cdots r_1 = r_n r_{n-1} \cdots r_{t+1} 0 1 \cdots 1$ . Then,  $|u| + 1$  is  $r_n r_{n-1} \cdots r_{t+1} 1 0 \cdots 0$  in binary and, by (1), the symbol  $(a_s, c) \notin \{a_t\} \times C$ . However, if  $s < t$ , then  $x_s^1$  goes to state  $q_{ns}$  under  $\{a_s\} \times C$ , while if  $s > t$ , then  $x_t^0$  goes to state  $q_{ns}$  under  $\{a_s\} \times C$ . In both cases, the non-secret state  $q_{ns}$  is in the next configuration of  $\mathcal{A}_\varphi$ , and hence  $\mathcal{A}_\varphi$  is CSO.

To prove that if  $\varphi$  is not satisfiable, then  $\mathcal{A}_\varphi$  is not CSO, we use the following claim.

▷ **Claim 13.** If  $\varphi$  is not satisfiable, there is a string  $w_\varphi \in Z_n$  such that the configuration of  $\mathcal{A}_\varphi$  after reading  $w_\varphi$  is  $\{x_n^1, x_{n-1}^1, \dots, x_1^1\} \cup \{q_s\}$ .

We now show that if  $\varphi$  is not satisfiable, then  $\mathcal{A}_\varphi$  is not CSO. To this end, we consider the string  $w_\varphi$  constructed in Claim 13, which we extend to a string from  $W_\varphi = Z_n \cdot (\{a_{n+1}\} \times C)$  by adding a symbol of the form  $\{a_{n+1}\} \times C$ . Since  $\varphi$  is not satisfiable, there is a clause  $c \in C$  that is not satisfied by the assignment of 1 to the variables; that is, there is  $c \notin \bigcup_{i=1}^n cl(x_i)$ . Then, the string  $w_\varphi(a_{n+1}, c)$  moves the automaton  $\mathcal{A}_\varphi$  from the configuration  $\{x_n^1, x_{n-1}^1, \dots, x_1^1\} \cup \{q_s\}$  to the configuration  $\{q_s\}$ , and hence  $\mathcal{A}_\varphi$  is not CSO. ◀

We now illustrate the construction.

► **Example 14.** For simplicity, we consider a 2-CNF formula

$$\varphi = (x_2 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_3)$$

with three variables  $x_1, x_2, x_3$  and five clauses  $c_1 = \{x_2, x_2\}$ ,  $c_2 = \{x_1, x_2\}$ ,  $c_3 = \{\neg x_1, x_3\}$ ,  $c_4 = \{\neg x_2, \neg x_3\}$ , and  $c_5 = \{x_3\}$ . The automaton  $\mathcal{A}_\varphi = (Q, \Sigma, \delta, \{q_s, x_1^0, x_2^0, x_3^0\})$  is depicted in Figure 2, where  $Q = \{q_s, q_{ns}\} \cup \{x_1^0, x_1^1, x_2^0, x_2^1, x_3^0, x_3^1\}$ ,  $\Sigma = \Gamma = \{a_1, a_2, a_3, a_4\} \times \{c_1, c_2, c_3, c_4, c_5\}$ , and  $q_s$  is the only secret state. Since  $\varphi$  is not satisfiable,  $\mathcal{A}_\varphi$  is not CSO; indeed, the string  $w = (a_1, c_1)(a_2, c_2)(a_1, c_5)(a_3, c_3)(a_1, c_1)(a_2, c_1)(a_1, c_4)(a_4, c_4)$  moves  $\mathcal{A}_\varphi$  to the configuration  $\{q_s\}$  consisting solely of the secret state, cf. Figure 3 depicting the reachable configurations of  $\mathcal{A}_\varphi$ .

On the other hand, if we consider the formula  $\varphi' = c_1 \wedge c_2 \wedge c_3 \wedge c_4$ , then  $\varphi'$  is satisfiable, and hence the NFA  $\mathcal{A}_{\varphi'}$  obtained from  $\mathcal{A}_\varphi$  by removing all transitions under symbols containing  $c_5$ , is CSO; it is visible from the reachable configurations of  $\mathcal{A}_{\varphi'}$  depicted in Figure 4.

The considered problems are all PSPACE-complete, and hence reducible to each other in polynomial time. However, this fact does not provide us with much information about the reductions. Even though some particular reductions have been discussed in the literature by Wu and Lafortune [43] and Balun and Masopust [5, 6], they are in most cases not suitable to prove lower bounds.

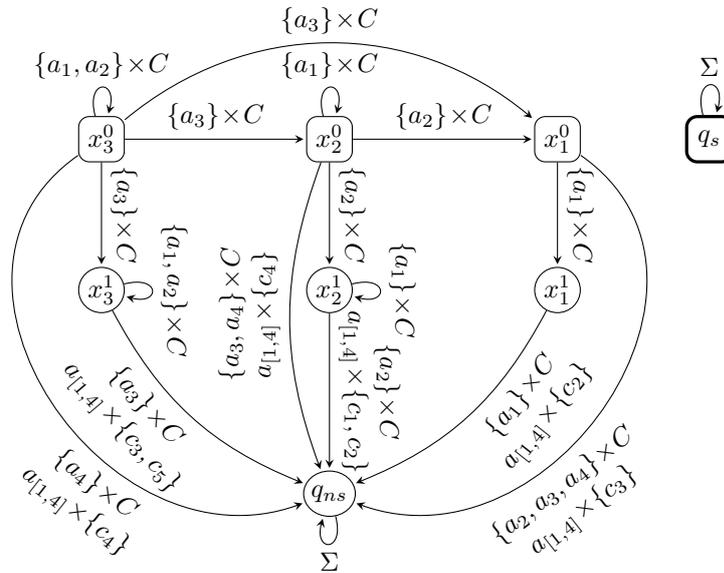


Figure 2 The NFA  $A_\varphi$  illustrating Theorem 8; the initial states are squared.

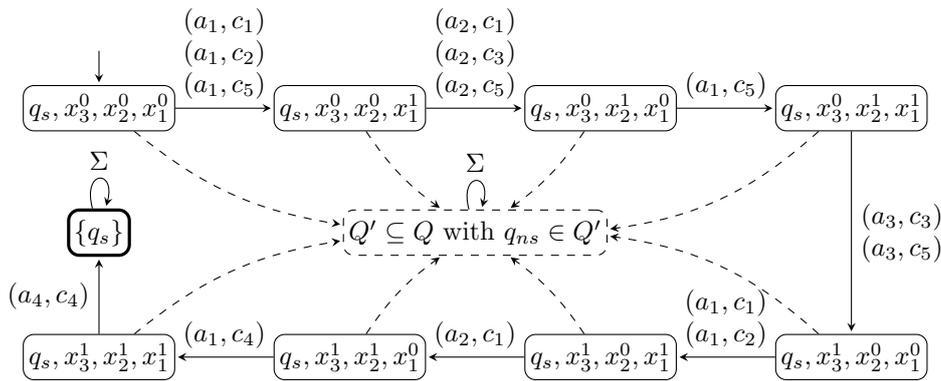


Figure 3 The configurations of  $A_\varphi$  – all undefined transitions go to the dashed middle state.

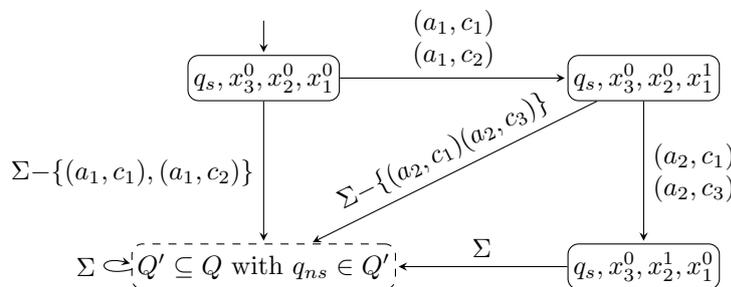


Figure 4 The configurations of  $A_{\varphi'}$ .

We now discuss the case of other types of opacity.

► **Corollary 15.** *Unless SETH fails, there is no algorithm deciding if a given  $n$ -state NFA is LBO/ISO/IFO/ $k$ -SO/INSO that runs in time  $O^*(2^{n/(2+\varepsilon)})$ , for any  $\varepsilon > 0$ .*

**Proof.** Consider the instance of CSO given by the NFA  $\mathcal{A}$  and the sets  $Q_S$  and  $Q_{NS}$  constructed in the proof of Theorem 8. Then,  $\mathcal{A}$  is CSO with respect to  $Q_S = \{q_s\}$  and  $Q_{NS} = Q - Q_S$  if and only if  $\mathcal{A}$  is LBO with respect to  $L_S = L_m(\mathcal{A}, q_s, q_s) = \Sigma^*$  and  $L_{NS} = L_m(\mathcal{A}, \{x_1^0, \dots, x_n^0\}, Q - \{q_s\})$ . Since the parts of  $\mathcal{A}$  corresponding to languages  $L_S$  and  $L_{NS}$  are disjoint, the instance of LBO may be encoded directly into  $\mathcal{A}$  by defining the corresponding states accepting the languages  $L_S$  and  $L_{NS}$ . Hence, the instance of LBO is of the same size as the instance of CSO. Therefore, if we solved the instance of LBO in time  $O^*(2^{(1-\delta)n})$ , we would also solve the instance of CSO in time  $O^*(2^{(1-\delta)n})$ .

Since there is no transition in  $\mathcal{A}$  from the sole secret state  $q_s$  to another state, the NFA  $\mathcal{A}$  is CSO if and only if  $\mathcal{A}$  is  $k$ -SO, for any  $k \in \mathbb{N} \cup \{\infty\}$ , and hence the result holds for  $k$ -SO as well as for INSO.

Furthermore, the NFA  $\mathcal{A}$  is CSO with respect to the sets  $Q_S$  and  $Q_{NS}$  if and only if  $\mathcal{A}$  is ISO with respect to the secret initial state  $I_S = \{q_s\}$  and non-secret initial states  $I_{NS} = \{x_1^0, \dots, x_n^0\}$ . Indeed, since  $L(\mathcal{A}, I_S) = \Sigma^*$ , the NFA  $\mathcal{A}$  is not CSO if and only if there is a string  $w \in \Sigma^*$  that moves  $\mathcal{A}$  from the initial configuration  $I_{NS}$  to the configuration  $\emptyset$ , which is if and only if  $\mathcal{A}$  is not ISO. As a result, solving ISO in time  $O^*(2^{(1-\delta)n})$  would solve CSO in time  $O^*(2^{(1-\delta)n})$ .

Finally, if all states of  $\mathcal{A}$  are accepting, then  $\mathcal{A}$  is ISO with respect to  $I_S = \{q_s\}$  and  $I_{NS} = \{x_1^0, \dots, x_n^0\}$  if and only if  $\mathcal{A}$  is IFO with respect to  $IQ_S = \{(q_s, q_s)\}$  and  $IQ_{NS} = I_{NS} \times Q$ ; hence, solving IFO in time  $O^*(2^{(1-\delta)n})$  would solve ISO in time  $O^*(2^{(1-\delta)n})$ . ◀

## 5 Lower Bounds under Exponential Time Hypothesis

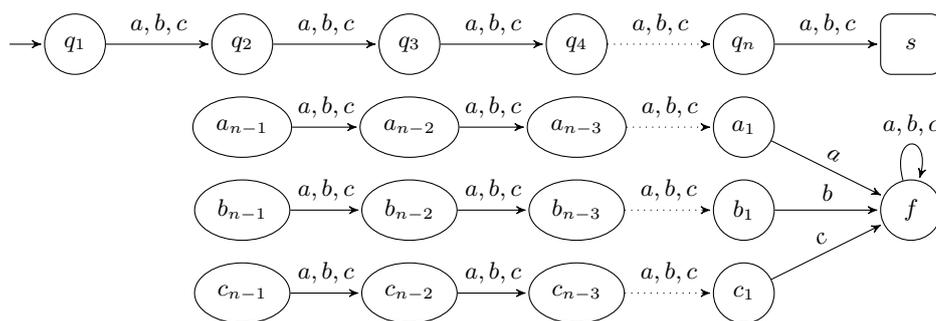
The number of symbols in the NFA constructed in Theorem 8 depends on the number of clauses in the instance of SAT. Since the standard binary encoding of symbols does not work under SETH, it is an open problem whether the results of Theorem 8 and Corollary 15 also hold for a fixed-sized alphabet.

Although we do not answer this question, we provide a lower bound for NFAs over a binary alphabet under ETH. Namely, we show that there is no algorithm solving the considered notions of opacity for such  $n$ -state NFAs that runs in time  $O^*(2^{o(n)})$ . We obtain the result by adjusting the construction of Fernau and Krebs [16], who showed that there is no algorithm solving the universality problem for  $n$ -state NFAs over a binary alphabet that runs in time  $O^*(2^{o(n)})$  unless ETH fails, and by using the observation of Cassez et al. [13] that universality can be reduced to opacity.

► **Theorem 16.** *Unless ETH fails, there is no algorithm deciding whether a given  $n$ -state NFA (over a binary alphabet) is CSO that runs in time  $O^*(2^{o(n)})$ .*

**Proof.** A 3-coloring of a graph  $G = (V, E)$  is a function  $\mu: V \rightarrow \{a, b, c\}$ . The coloring is proper if  $\mu(u) \neq \mu(v)$  whenever  $uv \in E$ . The 3-COLORING problem is to decide, given a graph  $G$ , whether there is a proper 3-coloring of  $G$ .

For a graph  $G$  with  $n$  vertices,  $V = \{v_1, v_2, \dots, v_n\}$ , and  $m$  edges, we construct an NFA  $\mathcal{A} = (Q, \Sigma, \delta, I)$ , where the states are  $Q = \{s, f\} \cup \{q_1, \dots, q_n\} \cup \{x_1, \dots, x_{n-1} \mid x \in \{a, b, c\}\}$ , the alphabet is  $\Sigma = \Gamma = \{a, b, c\}$ , the initial state is  $I = \{q_1\}$ , the secret state is  $Q_S = \{s\}$ , and the non-secret states are  $Q_{NS} = Q - Q_S$ . We define the transition function  $\delta$  as shown in Figure 5, and further extended it by adding three transitions  $(q_i, a, a_{j-i})$ ,  $(q_i, b, b_{j-i})$ , and  $(q_i, c, c_{j-i})$  for every edge  $v_i v_j \in E$  with  $i < j$ .



■ **Figure 5** The main part of the NFA  $\mathcal{A}$  resulting from the reduction of Theorem 16 without the transitions corresponding to the edges of  $G$ . The secret state is state  $s$ .

Intuitively, the coloring of  $G$  is encoded as a string  $w = c_1 \cdots c_n$  of length  $n$ , where  $c_i$  is the color of vertex  $i$ , and  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$  if and only if the non-secret state  $f$  is reachable under  $w$ ; indeed, the secret state  $s$  is reachable under every string of length  $n$ .

Fernau and Krebs [16] showed that  $\mathcal{A}$  ends up in state  $f$  under  $w$  if and only if  $w$  encodes a coloring that is not proper. Therefore, if  $w$  is a proper 3-coloring of  $G$ , then  $w$  does not move  $\mathcal{A}$  to state  $f$ ; that is, only the secret state  $s$  is reached under  $w$ , and hence  $\mathcal{A}$  is not CSO with respect to  $\{s\}$  and  $Q - \{s\}$ . On the other hand, if  $G$  does not have a proper 3-coloring, then every string of length  $n$  moves  $\mathcal{A}$  to both secret state  $s$  and non-secret state  $f$ ; that is,  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$ .

If  $G$  has  $n$  vertices and  $m$  edges, then  $\mathcal{A}$  has  $N = 4n - 1$  states and  $M = 12n + 3m - 12$  transitions. If there was an  $O^*(2^{o(N)})$ -time algorithm deciding CSO, we could reduce the instance of 3-COLORING to an instance of CSO in time  $O(N + M)$ , and solve CSO in time  $O^*(2^{o(N)})$ . Altogether, we could solve 3-COLORING in time  $O(N + M) + O^*(2^{o(N)}) = O^*(2^{o(n)})$ , which contradicts ETH. ◀

Similarly to the discussion in the previous section, to prove the lower bound for the other notions of opacity, it seems natural to combine the construction of the previous proof with the existing reductions among the notions [5, 6, 43]. However, most of the reductions result in too large, though polynomial, instances, and hence they are not suitable for our purposes. Therefore, new reductions are needed.

► **Corollary 17.** *Unless ETH fails, there is no algorithm deciding if a given  $n$ -state NFA (over a binary alphabet) is LBO/ISO/IFO/ $k$ -SO/INSO that runs in time  $O^*(2^{o(n)})$ .*

**Proof.** For the NFA  $\mathcal{A}$  of Theorem 16, we have  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$  if and only if  $\mathcal{A}$  is LBO with respect to  $L_S = L_m(\mathcal{A}, q_1, s)$  and  $L_{NS} = L_m(\mathcal{A}, q_1, Q - \{s\})$  [43]. If we could solve LBO in time  $O^*(2^{o(n)})$ , we could solve 3-COLORING in time  $O^*(2^{o(n)})$ .

Furthermore, since there is no transition from the sole secret state  $s$ , the automaton  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$  if and only if  $\mathcal{A}$  is  $k$ -SO with respect to  $\{s\}$  and  $Q - \{s\}$ , for any  $k \in \mathbb{N} \cup \{\infty\}$ . Therefore, the result holds for  $k$ -SO as well as for INSO.

Now, we take the NFA  $\mathcal{A}$  and add a copy of states  $q_1, q_2, \dots, q_n$ , denoted by  $q'_1, q'_2, \dots, q'_n$ , together with all transitions to states different from  $s$ , that is, we add  $(q'_i, x, p)$  for every transition  $(q_i, x, p)$  with  $p \neq s$ . We set the states  $q_1$  and  $q'_1$  initial, and denote the result by  $\mathcal{A}'$ . Then, the NFA  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$  if and only if  $\mathcal{A}'$  is ISO with respect to  $I_S = \{q_1\}$  and  $I_{NS} = \{q'_1\}$ . Indeed, if  $\mathcal{A}$  is CSO with respect to  $\{s\}$  and  $Q - \{s\}$ , then for every  $w$  moving  $\mathcal{A}$  to state  $s$ , there is  $w'$  moving  $\mathcal{A}$  to state  $f$ ; and so do the strings

$w$  and  $w'$  in  $\mathcal{A}'$ , which shows that  $\mathcal{A}'$  is ISO with respect to  $I_S = \{q_1\}$  and  $I_{NS} = \{q'_1\}$ . On the other hand, if  $\mathcal{A}$  is not CSO with respect to  $\{s\}$  and  $Q - \{s\}$ , then there is  $w$  moving  $\mathcal{A}$  only to state  $s$ , and hence  $w$  cannot be read by  $\mathcal{A}'$  from state  $q'_1$ , which shows that  $\mathcal{A}'$  is not ISO with respect to  $I_S = \{q_1\}$  and  $I_{NS} = \{q'_1\}$ . If we could solve ISO in time  $O^*(2^{o(n)})$ , we could solve 3-COLORING in time  $O^*(2^{o(n)})$  by reducing it to ISO and solving ISO in time  $O^*(2^{o(N+n)}) = O^*(2^{o(n)})$ , for  $N = 4n - 1$ .

If we in addition set the states  $s$  and  $f$  accepting, then  $\mathcal{A}'$  is ISO with respect to  $I_S = \{q_1\}$  and  $I_{NS} = \{q'_1\}$  if and only if  $\mathcal{A}'$  is IFO with respect to  $IQ_S = \{(q_1, s)\}$  and  $IQ_{NS} = \{(q'_1, f)\}$ , and hence if we could solve IFO in time  $O^*(2^{o(n)})$ , we could solve 3-COLORING in time  $O^*(2^{o(n)})$ . ◀

## 6 Discussion and Conclusions

We showed that if the strong exponential time hypothesis holds true, then, for any  $c > 2$ , there are no algorithms deciding various types of opacity in time  $O^*(2^{n/c})$ . Therefore, the current algorithms cannot be significantly improved.

More precisely, the results say that there are no algorithms deciding various types of opacity in time  $O^*(\sqrt{2}^n) = O^*(1.414213562^n)$ . However, the results admit the existence of algorithms deciding opacity in time  $O^*(1.5^n)$ . Whether such algorithms exist or whether the current lower bounds can be strengthened remains an open problem.

The construction used in the proof of Theorem 8 can be utilized to improve the conditional lower bound of deciding universality for NFAs. The universality problem for NFAs asks whether, given an NFA, the NFA accepts all strings over its alphabet. If we set the only secret state  $q_s$  of the NFA  $\mathcal{A}$  of Theorem 8 to be non-accepting and all the other states to be accepting, we obtain an NFA that is universal if and only if the automaton  $\mathcal{A}$  is CSO with respect to  $\{q_s\}$  and  $Q - \{q_s\}$ . We thus have the following consequence improving the result of Fernau and Krebs [16].

► **Corollary 18.** *Unless SETH fails, there is no algorithm deciding whether a given  $n$ -state NFA is universal that runs in time  $O^*(2^{n/(2+\varepsilon)})$ , for any  $\varepsilon > 0$ .* ◀

Consequently, we immediately have the following result.

► **Corollary 19.** *Given two NFAs  $\mathcal{A}_1$  and  $\mathcal{A}_2$  with  $n_1$  and  $n_2$  states, respectively, let  $n = \max(n_1, n_2)$ . Unless SETH fails, there is no algorithm deciding whether  $L_m(\mathcal{A}_1) \subseteq L_m(\mathcal{A}_2)$  in time  $O^*(2^{n/(2+\varepsilon)})$ , and there is no algorithm deciding whether  $L_m(\mathcal{A}_1) = L_m(\mathcal{A}_2)$  in time  $O^*(2^{n/(2+\varepsilon)})$ , for any  $\varepsilon > 0$ .* ◀

We left the question whether Theorem 8 also holds for NFAs over a fixed-size alphabet open. Although we did not answer this question, we showed that ETH implies the non-existence of sub-exponential-time algorithms deciding various types of opacity over a binary alphabet.

Inspecting Table 1, the reader may notice quite a large gap between the lower and upper bounds for the verification of IFO without any restrictions on the form of non-secret pairs. To improve the upper bound or to (conditionally) show that no such improvements are possible is a challenging open problem.

It is worth noticing that the construction in the proof of Corollary 15 produces an instance of a special case of the problem where the non-secret pairs are of the form  $IQ_{NS} = I_{NS} \times F_{NS}$ , and hence the special case is tight under the strong exponential time hypothesis.

---

**References**

---

- 1 Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving secrecy under refinement. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *International Colloquium on Automata, Languages and Programming, ICALP 2006, Venice, Italy, July 10-14, 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006. doi:10.1007/11787006\_10.
- 2 Peter R. J. Asveld and Anton Nijholt. The inclusion problem for some subclasses of context-free languages. *Theoretical Computer Science*, 230(1-2):247–256, 2000. doi:10.1016/S0304-3975(99)00113-9.
- 3 Éric Badouel, Marek A. Bednarczyk, Andrzej M. Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, 2007. doi:10.1007/s10626-007-0020-5.
- 4 Jiří Balun and Tomáš Masopust. On opacity verification for discrete-event systems. In *IFAC World Congress, Berlin, Germany, July 11-17, 2020*, pages 2105–2110, 2020. doi:10.1016/j.ifacol.2020.12.2524.
- 5 Jiří Balun and Tomáš Masopust. Comparing the notions of opacity for discrete-event systems. *Discrete Event Dynamic Systems*, 31(4):553–582, 2021. doi:10.1007/s10626-021-00344-2.
- 6 Jiří Balun and Tomáš Masopust. On transformations among opacity notions. In *IEEE International Conference on Systems, Man, and Cybernetics, SMC 2022, Prague, Czech Republic, October 9-12, 2022*, pages 3012–3017. IEEE, 2022. doi:10.1109/SMC53654.2022.9945608.
- 7 Jiří Balun, Tomáš Masopust, and Petr Osička. Speed me up if you can: Conditional lower bounds on opacity verification. *CoRR*, abs/2304.09920, 2023. doi:10.48550/arXiv.2304.09920.
- 8 Eike Best, Philippe Darondeau, and Roberto Gorrieri. On the decidability of non interference over unbounded Petri nets. In Konstantinos Chatzikokolakis and Véronique Cortier, editors, *International Workshop on Security Issues in Concurrency, SecCo 2010, Paris, France, August 30, 2010*, volume 51 of *EPTCS*, pages 16–33, 2010. doi:10.4204/EPTCS.51.2.
- 9 Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008. doi:10.1007/s10207-008-0058-x.
- 10 Jeremy W. Bryans, Maciej Koutny, and Peter Y. A. Ryan. Modelling dynamic opacity using petri nets with silent actions. In Theodosios Dimitrakos and Fabio Martinelli, editors, *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, Toulouse, France, August 22-27, 2004*, volume 173 of *IFIP*, pages 159–172. Springer, 2004. doi:10.1007/0-387-24098-5\_12.
- 11 Jeremy W. Bryans, Maciej Koutny, and Peter Y.A. Ryan. Modelling opacity using Petri nets. *Electronic Notes in Theoretical Computer Science*, 121:101–115, 2005. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004). doi:10.1016/j.entcs.2004.10.010.
- 12 Nadia Busi and Roberto Gorrieri. Structural non-interference in elementary and trace nets. *Mathematical Structures in Computer Science*, 19(6):1065–1090, 2009. doi:10.1017/S0960129509990120.
- 13 Franck Cassez, Jérémy Dubreil, and Hervé Marchand. Synthesis of opaque systems with static and dynamic masks. *Formal Methods in System Design*, 40(1):88–115, 2012. doi:10.1007/s10703-012-0141-9.
- 14 Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in P2P anonymity systems. In *Workshop on Economics of Peer-to-Peer Systems, P2P Econ 2003, Berkeley, CA, USA, June 5-6, 2003*, 2003.

- 15 Jeremy Dubreil, Philippe Darondeau, and Herve Marchand. Opacity enforcing control synthesis. In *International Workshop on Discrete Event Systems, WODES 2008, Gothenburg, Sweden, May 28-30, 2008*, pages 28–35. IEEE, 2008. doi:10.1109/WODES.2008.4605918.
- 16 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017. doi:10.3390/a10010024.
- 17 R. Focardi and R. Gorrieri. A taxonomy of trace-based security properties for CCS. In *Computer Security Foundations Workshop VII, Franconia, NH, USA, June 14-16, 1994*, pages 126–136, 1994. doi:10.1109/CSFW.1994.315941.
- 18 Rômulo Meira Góes, Blake C. Rawlings, Nicholas Recker, Gregory Willett, and Stéphane Lafortune. Demonstration of indoor location privacy enforcement using obfuscation. *IFAC-PapersOnLine*, 51(7):145–151, 2018. doi:10.1016/j.ifacol.2018.06.293.
- 19 N. B. Hadj-Alouane, S. Lafrance, Feng Lin, J. Mullins, and M. M. Yeddes. On the verification of intransitive noninterference in multilevel security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(5):948–958, 2005. doi:10.1109/TSMCB.2005.847749.
- 20 Christoforos N. Hadjicostis. *Estimation and Inference in Discrete Event Systems: A Model-Based Approach with Finite Automata*. Communications and Control Engineering. Springer Cham, 2020. doi:10.1007/978-3-030-30821-6.
- 21 Timon Hertli. 3-SAT faster and simpler - unique-sat bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014. doi:10.1137/120868177.
- 22 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Pearson international edition. Addison-Wesley, third edition, 2007.
- 23 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.
- 24 Romain Jacob, Jean-Jacques Lesage, and Jean-Marc Faure. Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41:135–146, 2016. doi:10.1016/j.arcontrol.2016.04.015.
- 25 Feng Lin. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems*, 4(2):197–212, 1994. doi:10.1007/BF01441211.
- 26 Feng Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011. doi:10.1016/j.automatica.2011.01.002.
- 27 Feng Lin and Walter Murray Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988. doi:10.1016/0020-0255(88)90001-1.
- 28 Tomáš Masopust. Complexity of deciding detectability in discrete event systems. *Automatica*, 93:257–261, 2018. doi:10.1016/j.automatica.2018.03.077.
- 29 Tomáš Masopust and Xiang Yin. Complexity of detectability, opacity and A-diagnosability for modular discrete event systems. *Automatica*, 101:290–295, 2019. doi:10.1016/j.automatica.2018.12.019.
- 30 Laurent Mazaré. Decidability of opacity with non-atomic keys. In Theodosios Dimitrakos and Fabio Martinelli, editors, *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, Toulouse, France, August 22-27, 2004*, volume 173 of *IFIP*, pages 71–84. Springer, 2004. doi:10.1007/0-387-24098-5\_6.
- 31 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.
- 32 Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998. doi:10.1145/290163.290168.
- 33 Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003. doi:10.1109/JSAC.2002.806121.

- 34 Anooshiravan Saboori. *Verification and enforcement of state-based notions of opacity in discrete event systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.
- 35 Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of infinite-step opacity and complexity considerations. *IEEE Transactions on Automatic Control*, 57(5):1265–1269, 2012. doi:10.1109/TAC.2011.2173774.
- 36 Anooshiravan Saboori and Christoforos N. Hadjicostis. Verification of initial-state opacity in security applications of discrete event systems. *Information Sciences*, 246:115–132, 2013. doi:10.1016/j.ins.2013.05.033.
- 37 Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995. doi:10.1109/9.412626.
- 38 Steve A. Schneider and Abraham Sidiropoulos. CSP and anonymity. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *European Symposium on Research in Computer Security, ESORICS 1996, Rome, Italy, September 25-27, 1996*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer, 1996. doi:10.1007/3-540-61770-1\_38.
- 39 Shaolong Shu, Feng Lin, and Hao Ying. Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12):2356–2359, 2007. doi:10.1109/TAC.2007.910713.
- 40 N. J. A. Sloan. The on-line encyclopedia of integer sequences (OEIS), 2023. A123121. URL: <https://oeis.org/A123121>.
- 41 N. J. A. Sloan. The on-line encyclopedia of integer sequences (OEIS), 2023. A001511. URL: <https://oeis.org/A001511>.
- 42 Andrew Wintenberg, Matthew Blischke, Stéphane Lafortune, and Necmiye Ozay. A dynamic obfuscation framework for security and utility. In *ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2022, Milano, Italy, May 4-6, 2022*, pages 236–246. IEEE, 2022. doi:10.1109/ICCPS54341.2022.00028.
- 43 Yi-Chin Wu and Stéphane Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems*, 23(3):307–339, 2013. doi:10.1007/s10626-012-0145-z.
- 44 Yi-Chin Wu, Vasumathi Raman, Blake C. Rawlings, Stéphane Lafortune, and Sanjit A. Seshia. Synthesis of obfuscation policies to ensure privacy and utility. *Journal of Automated Reasoning*, 60(1):107–131, 2018. doi:10.1007/s10817-017-9420-x.
- 45 Xiang Yin and Stéphane Lafortune. A new approach for the verification of infinite-step and K-step opacity using two-way observers. *Automatica*, 80:162–171, 2017. doi:10.1016/j.automatica.2017.02.037.
- 46 Aris Zakynthinos and E. Stewart Lee. A general theory of security properties. In *IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 4-7, 1997*, pages 94–102. IEEE Computer Society, 1997. doi:10.1109/SECPRI.1997.601322.



# Separating Automatic Relations

Pablo Barceló   

Institute for Mathematical and Computational Engineering, Universidad Católica de Chile & CENIA & IMFD, Santiago, Chile

Diego Figueira   

Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France

Rémi Morvan   

Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR5800, F-33400 Talence, France

---

## Abstract

We study the separability problem for automatic relations (i.e., relations on finite words definable by synchronous automata) in terms of recognizable relations (i.e., finite unions of products of regular languages). This problem takes as input two automatic relations  $R$  and  $R'$ , and asks if there exists a recognizable relation  $S$  that contains  $R$  and does not intersect  $R'$ . We show this problem to be undecidable when the number of products allowed in the recognizable relation is fixed. In particular, checking if there exists a recognizable relation  $S$  with at most  $k$  products of regular languages that separates  $R$  from  $R'$  is undecidable, for each fixed  $k \geq 2$ . Our proofs reveal tight connections, of independent interest, between the separability problem and the finite coloring problem for automatic graphs, where colors are regular languages.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Regular languages

**Keywords and phrases** Automatic relations, recognizable relations, separability, finite colorability

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.17

**Related Version** *Full Version*: <https://arxiv.org/abs/2305.08727>

**Funding** Figueira and Morvan are partially supported by ANR QUID, grant ANR-18-CE400031. Barceló is funded by ANID – Millennium Science Initiative Program – Code ICN17002 and by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID.

 This pdf contains internal links: clicking on a notion leads to its *definition*.<sup>1</sup>

## 1 Introduction

**Context.** The study of classes of relations on words has become an important topic in language theory [12, 23, 5, 13, 9], and also in areas such as databases and verification where they are used to build expressive languages. For instance, classes of relations of this kind are relevant for querying strings over relational databases [3], comparing paths in graph databases [2], or defining string constraints for model checking [20]. The most studied such classes include *recognizable*, *automatic*, and *rational* relations, each one of the latter two strictly extending the previous one. *Rational relations* are those definable by multi-head automata, with heads possibly moving asynchronously; automatic relations are rational relations that are accepted by multi-head automata whose heads are forced to move synchronously; and recognizable relations correspond to finite unions of products of regular languages (or, equivalently, to languages recognized via finite monoids, by Mezei’s Theorem). By definition, all of these classes coincide with the class of regular languages when restricted to unary relations.

---

<sup>1</sup> This result was achieved by using the `knowledge` package and its companion tool `knowledge-clustering`.



Prior work has focused on the *REC-DEFINABILITY PROBLEM*, which takes as input an  $n$ -ary rational relation  $R$  and asks whether it is equivalent to a recognizable relation  $\bigcup_i L_{i,1} \times \cdots \times L_{i,n}$ , where each  $L_{i,j}$  is a regular language. Intuitively, the problem asks whether the different components of the rational relation  $R$  are almost independent of one another. The study of REC-DEFINABILITY is relevant since relations enjoying this property are often amenable to some analysis including, e.g., abstract interpretations in program verification, variable elimination in constraint logic programming, and query processing over constraint databases (see the introduction of [1] for a thorough discussion on this topic).

In general, REC-DEFINABILITY of rational relations is undecidable, but it becomes decidable for two important subclasses: deterministic rational relations and automatic relations. For deterministic rational relations, REC-DEFINABILITY has been shown to be decidable in double-exponential time for binary relations by Valiant [27] – improving Stearns’s triple-exponential bound [25]. The decidability result was later extended to relations of arbitrary arity by Carton, Choffrut and Grigorieff [8, Theorem 3.7]. For automatic relations, the decidability of REC-DEFINABILITY can be obtained by a simple reduction to the problem of checking whether a finite automaton recognizes an infinite language [21] – which is decidable via a standard reachability argument. The precise complexity of the problem, however, was only recently pinned down. By applying techniques based on Ramsey Theorem over infinite graphs, it was shown that REC-DEFINABILITY of automatic relations is PSPACE-complete when relations are specified by non-deterministic automata [1, Theorem 1] [4, Corollary 2.9].

On the other hand, much less is known about the *REC-SEPARABILITY PROBLEM*, which takes two  $n$ -ary rational relations  $R, R' \subseteq \mathbb{A}^* \times \mathbb{A}^*$  and checks whether there is a recognizable relation  $S = \bigcup_i L_{i,1} \times \cdots \times L_{i,n}$  with  $R \subseteq S$  and  $R' \cap S = \emptyset$ . In other words, this problem asks whether we can *overapproximate*  $R$  with a recognizable relation  $S$  that is constrained not to intersect with  $R'$ . Separability problems of this kind abound in theoretical computer science, in particular in formal language theory where they have gained a lot of attention over the last few years – see, e.g., [24, 16, 11, 10].

As for definability, the REC-SEPARABILITY PROBLEM for rational relations is in general undecidable. In this paper we focus on the separability problem for automatic relations, that is, the restriction of the REC-SEPARABILITY PROBLEM defined above to the case when both  $R$  and  $R'$  are automatic relations. Notice that when  $R'$  is the complement of  $R$  this problem boils down to REC-DEFINABILITY. However, REC-SEPARABILITY for automatic relations is more general than REC-DEFINABILITY, and to this day it is unknown whether it is decidable.

**Main contributions and technical approach.** While we do not solve the separability problem for automatic relations, we report on some significant progress in our understanding of the problem. We start by establishing a tight connection between REC-SEPARABILITY and the colorability problem for “automatic graphs”, which may shed some light on the difficulty of the former problem. An automatic graph [7, 14, 17, 18] is an infinite graph defined on a regular set of finite words, whose edge set is described by a binary automatic relation. The REGULAR COLORABILITY PROBLEM is then the problem of checking if a given automatic graph is finitely colorable, with the restriction that each color forms a regular language. Concretely, we show that the REC-SEPARABILITY PROBLEM for binary automatic relations is equivalent, under polynomial time reductions, to the REGULAR COLORABILITY PROBLEM. Moreover, we introduce a hierarchy  $(k\text{-REC})_{k>0}$  of recognizable relations so that the coloring problem, when restricted to  $k > 0$  colors – called  $k$ -REGULAR COLORABILITY PROBLEM – reduces to the separability problem by relations of  $k$ -REC. Concretely:

► **Theorem 3.1.** *There are polynomial-time reductions:*

1. *from the REC-SEPARABILITY PROBLEM to the REGULAR COLORABILITY PROBLEM;*
2. *from the REGULAR COLORABILITY PROBLEM to the REC-SEPARABILITY PROBLEM; and*
3. *from the  $k$ -REGULAR COLORABILITY PROBLEM to the  $k$ -REC-SEPARABILITY PROBLEM, for every  $k > 0$ .*

*Further, the last two reductions are so that the second relation in the instance of the SEPARABILITY PROBLEM is the identity  $Id$ .*

The REGULAR COLORABILITY PROBLEM seems challenging, and in particular we lack tools for establishing that an automatic graph is finitely colorable; let alone checking that said colors define regular sets. On the other hand, it is easy to see that the  $k$ -REGULAR COLORABILITY PROBLEM is undecidable for each fixed  $k > 1$  if we lift the restriction that colors define regular sets, i.e., checking if an automatic graph admits a  $k$ -coloring – this has been proved in an unpublished thesis by Köcher [15, Proposition 6.5]. To be more precise, the problem is even co-recursively enumerable-complete<sup>2</sup>. We establish that this undecidability holds even with the restriction on colors being regular sets:

► **Theorem 4.4.** *The  $k$ -REGULAR COLORABILITY PROBLEM on automatic graphs is undecidable, for every  $k \geq 2$ . More precisely, the problem is recursively enumerable-complete. This holds also for connected automatic graphs.*

Note that the definitions of  $k$ -REGULAR COLORABILITY PROBLEM and  $k$ -COLORABILITY PROBLEM look similar, and are both undecidable, but the former is RE-complete while the latter is CORE-complete.

By reduction from the  $k$ -REGULAR COLORABILITY PROBLEM we obtain an important consequence for our separability problem: It is undecidable to check if two automatic relations can be separated by a recognizable relation defined by a *fixed* number of unions of products of regular languages. More specifically, fix  $k > 0$  and define  $k$ -PROD as the class of recognizable relations of the form  $S = \bigcup_{1 \leq i \leq k} L_{i,1} \times \cdots \times L_{i,n}$  – this hierarchy is intertwined with the  $(k\text{-REC})_{k>0}$  hierarchy introduced previously. We show that the  $k$ -PROD-SEPARABILITY PROBLEM, i.e., the problem of checking separability for binary automatic relations  $R$  and  $R'$  in terms of a recognizable relation  $S$  in the class  $k$ -PROD, is undecidable for any  $k \geq 2$ .

► **Theorem 5.6.** *The  $k$ -PROD-SEPARABILITY PROBLEM is undecidable, for every  $k \geq 2$ .*

At this point, a natural question is whether our choice of restricting the study to the class  $k$ -PROD, for fixed  $k > 1$ , is not too strong, in the sense that it turns undecidable not only the separability but also the *definability* problem for automatic relations. We show that this is not the case; in fact, by using a simple adaptation of the proof techniques in [1] we can show that the problem of checking if an automatic relation can be expressed as a relation in  $k$ -PROD, for any fixed  $k > 0$ , is decidable in single-exponential time:

► **Corollary 6.4.** *The  $k$ -PROD-DEFINABILITY PROBLEM is decidable, for every  $k > 0$ .*

**Remark.** For simplicity, we focus on binary automatic relations only. Extending the decidability results to  $n$ -ary automatic relations, for  $n > 2$  is direct by applying tools in [1].

---

<sup>2</sup> The upper bound follows from De Bruijn–Erdős Theorem.

## 2 Preliminaries

**Automatic and recognizable relations.** Let  $\mathbb{A}$  be a finite alphabet. We write  $\mathbb{A}_\perp$  for the extension of  $\mathbb{A}$  with a fresh symbol  $\perp$ . Given a pair  $(w_1, w_2) \in \mathbb{A}^* \times \mathbb{A}^*$ , we write  $w_1 \otimes w_2$  for the word over alphabet  $\mathbb{A}_\perp \times \mathbb{A}_\perp$  that is obtained as follows: first, padding the shorter word with  $\perp$ 's until both words are of the same length, and then reading the two words synchronously as if they were a single word over a binary alphabet. For example, if  $w_1 = aaba$  and  $w_2 = aa$ , then  $w_1 \otimes w_2 = (a, a)(a, a)(b, \perp)(a, \perp)$ . For any relation  $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ , let us write  $\otimes R$  to denote the set

$$\otimes R \hat{=} \{u \otimes v \mid (u, v) \in R\} \subseteq (\mathbb{A}_\perp \times \mathbb{A}_\perp)^*.$$

We then have the following:

- $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$  is an *automatic relation* iff  $\otimes R$  is a regular language;
- $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$  is a *recognizable relation* iff  $R = \bigcup_{i=1}^n A_i \times B_i$ , where  $n \in \mathbb{N}$  and all the  $A_i$ 's and  $B_i$ 's are regular languages over  $\mathbb{A}$ .

We denote by REC the class of all recognizable relations.

► **Example 2.1.** For any fixed constant  $c > 0$ , the relation  $R$  composed by all pairs of words of the form  $(a^n, a^{n+c})$ , for  $n \geq 0$ , is automatic. In turn,  $R$  is not recognizable. An example of a non-automatic relation is the one consisting of all pairs of the form  $(a^n, a^{d \cdot n})$ , for  $n > 0$ , for any constant  $d > 1$ . ◀

**Separability.** Let  $R$  and  $R'$  be automatic relations over an alphabet  $\mathbb{A}$ . A recognizable relation  $S$  over  $\mathbb{A}$  *separates*  $R$  from  $R'$  if  $R \subseteq S$  and  $R' \cap S = \emptyset$ .

► **Example 2.2.** Consider the automatic relations  $R = \{(a^n, a^{n+1}) \mid n \geq 0\}$  and  $R' = \{(a^n, a^{n+2}) \mid n \geq 0\}$ . They are separable by the recognizable relation

$$S = (A_{\text{even}} \times A_{\text{odd}}) \cup (A_{\text{odd}} \times A_{\text{even}}),$$

where  $A_{\text{even}}$  and  $A_{\text{odd}}$  are the regular languages  $(aa)^*$  and  $a(aa)^*$ , respectively. ◀

We study the following separability problem, for a class  $\mathcal{C}$  of recognizable relations.

**Problem:**  $\mathcal{C}$ -SEPARABILITY PROBLEM  
**Input:** Automatic relations  $R$  and  $R'$  over  $\mathbb{A}$   
**Question:** Is there a recognizable relation in  $\mathcal{C}$  over  $\mathbb{A}$  that separates  $R$  from  $R'$ ?

We also consider the  $\mathcal{C}$ -DEFINABILITY PROBLEM, which takes as input an automatic relation  $R$  and asks if there is a recognizable relation  $S$  in  $\mathcal{C}$  with  $S = R$ . It is easy to see that the  $\mathcal{C}$ -DEFINABILITY PROBLEM corresponds to an instance of the  $\mathcal{C}$ -SEPARABILITY PROBLEM.

▷ **Fact 2.3.** For any class  $\mathcal{C}$  of recognizable relations, the  $\mathcal{C}$ -DEFINABILITY PROBLEM is Turing-reducible to the  $\mathcal{C}$ -SEPARABILITY PROBLEM.

**Proof.** Reduce an instance  $R$  of the DEFINABILITY PROBLEM to the instance  $(R, (\mathbb{A}^* \times \mathbb{A}^*) \setminus R)$  of the SEPARABILITY PROBLEM. ◀

The following is known regarding the complexity of the REC-definability problem.

► **Proposition 2.4** ([1, Theorem 1]). *The REC-DEFINABILITY PROBLEM for automatic relations specified by non-deterministic automata is PSPACE-complete.*

**Automatic graphs.** Let  $L$  be a language of finite words over  $\mathbb{A}$ , and  $R \subseteq L \times L$  binary relation over  $\mathbb{A}$ . They naturally define a directed graph  $G = \langle L, R \rangle$ , *i.e.*, the nodes of  $G$  are the words over  $L$  and there is an edge in  $G$  from word  $u$  to word  $v$  iff  $(u, v) \in R$ . An *automatic graph* is a graph of the form  $\langle L, R \rangle$ , for  $R$  an automatic relation and  $L$  a regular language<sup>3</sup>. A  $k$ -*coloring* of  $\langle L, R \rangle$  is a partition of  $L$  into  $k$  sets  $V_1, \dots, V_k$  such that  $(V_i \times V_i) \cap E = \emptyset$  for every  $i$ .

► **Example 2.5.** Consider again the automatic relation  $R = \{(a^n, a^{n+c}) \mid n \geq 0\}$ , where  $c > 0$  is a fixed constant. The graph  $\langle a^*, R \rangle$  is formed by a disjoint union of  $c$  infinite directed paths, and thus it is 2-colorable. ◻

A  $k$ -*regular coloring* of an automatic graph is a  $k$ -coloring whose colors  $(V_i)_{1 \leq i \leq k}$  are regular languages. A *regular coloring* is a  $k$ -regular coloring for some  $k$ .

► **Example 2.6.** The automatic graph  $\langle a^*, R \rangle$  from Example 2.5 is 2-regular colorable. In fact, it suffices to define color  $V_1$  as having every word of the form  $a^n$  with  $n \equiv i \pmod{2c}$ , for  $i \in [0, c-1]$ , and  $V_2 = \mathbb{A}^* \setminus V_1$ . ◻

The  $k$ -REGULAR COLORABILITY PROBLEM is the problem of whether a given automatic graph has a  $k$ -regular coloring. The REGULAR COLORABILITY PROBLEM is the problem of whether a given automatic graph has a regular coloring.

### 3 Separability is Equivalent to Regular Colorability

We start by showing that the SEPARABILITY PROBLEM in terms of arbitrary recognizable relations is equivalent, under polynomial time reductions, to the REGULAR COLORABILITY PROBLEM. To make our statement precise, we need some terminology introduced below. Let  $k$ -REC be the class of languages expressed by unions of products of  $k$  regular languages which form a partition, that is (in the binary case), relations of the form  $(L_{i_1} \times L_{j_1}) \cup \dots \cup (L_{i_\ell} \times L_{j_\ell})$ , with  $i_1, j_1, \dots, i_\ell, j_\ell \in [1, k]$ , for some regular partition  $L_1, \dots, L_k$  of  $\mathbb{A}^*$  and  $\ell \in \mathbb{N}$ . Note that  $\text{REC} = \bigcup_k k\text{-REC}$ . Let us denote by  $Id$  the identity relation (on any implicit alphabet). Observe that  $Id$  is automatic but not recognizable.

► **Theorem 3.1.** *There are polynomial-time reductions:*

1. from the REC-SEPARABILITY PROBLEM to the REGULAR COLORABILITY PROBLEM;
2. from the REGULAR COLORABILITY PROBLEM to the REC-SEPARABILITY PROBLEM; and
3. from the  $k$ -REGULAR COLORABILITY PROBLEM to the  $k$ -REC-SEPARABILITY PROBLEM, for every  $k > 0$ .

Further, the last two reductions are so that the second relation in the instance of the SEPARABILITY PROBLEM is the identity  $Id$ .

**Proof.** We start with the last two reductions. Given an automatic graph  $\langle L, E \rangle$  over an alphabet  $\mathbb{A}$ , consider the instance  $R_1, R_2$  for the REC-SEPARABILITY PROBLEM, where  $R_1 = E$  and  $R_2 = Id$ . If  $\langle L, E \rangle$  is  $k$ -regular colorable via the coloring  $V_1, \dots, V_k$  then the  $k$ -REC relation  $\bigcup_{i \neq j} V_i \times V_j$  separates  $R_1$  and  $R_2$ . Conversely, if a  $k$ -REC relation  $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$  on the partition  $V_1 \dot{\cup} \dots \dot{\cup} V_k = \mathbb{A}^*$  separates  $R_1$  and  $R_2$ , then  $\bigcup_{i \neq j} V_i \times V_j$  also separates  $R_1$  and  $R_2$ , and this implies that  $V_1, \dots, V_k$  is a  $k$ -coloring for  $\langle \mathbb{A}^*, E \rangle$ .

<sup>3</sup> Note that an automatic graph can contain self-loops. However, since the presence of such an edge prevent the graph from being  $k$ -colorable for any  $k \geq 0$ , all our examples will be self-loop-free.

## 17:6 Separating Automatic Relations

For the first reduction, let us introduce some terminology. Given two relations  $R_1, R_2$  over  $\mathbb{A}^*$ , say that  $u \in \mathbb{A}^*$  is *compatible* with  $u' \in \mathbb{A}^*$  when for all words  $v \in \mathbb{A}^*$ :

$$\begin{aligned} (\text{COMP}_\ell): (u, v) \in R_1 \Rightarrow (u', v) \notin R_2, & \quad (\text{COMP}_r): (v, u) \in R_1 \Rightarrow (v, u') \notin R_2, \\ (\text{COMP}'_\ell): (u', v) \in R_1 \Rightarrow (u, v) \notin R_2 & \quad \text{and} \quad (\text{COMP}'_r): (v, u') \in R_1 \Rightarrow (v, u) \notin R_2. \end{aligned}$$

Define the *incompatibility graph*  $\mathcal{I}nc_{R_1, R_2}$  as the graph whose vertices are all words of  $\mathbb{A}^*$ , and with an edge from  $u$  to  $v$  whenever  $u$  is not compatible with  $v$ . Note that  $\mathcal{I}nc_{R, Id}$  is exactly the graph  $\langle \mathbb{A}^*, R \rangle$ . For a less trivial example of an incompatibility graph, see the full version.

► **Lemma 3.2.** *If  $R_1$  and  $R_2$  are automatic, then so is  $\mathcal{I}nc_{R_1, R_2}$ . Moreover, we can build an automaton for  $\mathcal{I}nc_{R_1, R_2}$  in polynomial time in the size of the automata for  $R_1$  and  $R_2$ .*

Given an instance  $(R_1, R_2)$  of the SEPARABILITY PROBLEM, we reduce it to the REGULAR COLORABILITY PROBLEM on its incompatibility graph  $\mathcal{I}nc_{R_1, R_2}$ .

**Left-to-right implication:** Assume that there exists  $S$  in  $k$ -REC that separates  $R_1$  from  $R_2$ . Then  $S$  can be written as  $(A_{i_1} \times A_{j_1}) \cup \dots \cup (A_{i_\ell} \times A_{j_\ell})$ , where  $(A_1, \dots, A_k)$  is a partition of  $\mathbb{A}^*$  in  $k$  regular languages. We define the color of a word  $u \in \mathbb{A}^*$  as the unique  $i \in \llbracket 1, k \rrbracket$  s.t.  $u \in A_i$ . In other words, the coloring is simply  $(A_1, \dots, A_k)$ .

This is indeed a proper coloring: if  $u$  and  $u'$  have the same color, we claim that  $u$  is compatible with  $u'$ . Indeed, take any  $v \in \mathbb{A}^*$ : if  $(u, v) \in R_1$ , then  $(u, v) \in S$ , so  $(u, v) \in A_{i_m} \times A_{j_m}$  for some  $m$ . But since  $u$  has the same color as  $u'$ , the fact that  $u \in A_{i_m}$  implies  $u' \in A_{i_m}$ , and hence  $(u', v) \in A_{i_m} \times A_{j_m} \subseteq S$ . But  $S$  separates  $R_1$  from  $R_2$ , and therefore  $(u', v) \notin R_2$ . This tells us that  $(\text{COMP}_\ell)$  holds. The other conditions hold by symmetry. We conclude that  $(A_1, \dots, A_k)$  defines a proper coloring of  $\mathcal{I}nc_{R_1, R_2}$ , and this coloring, with  $k$  colors, is regular since the  $A_i$ 's are regular languages by definition.

**Right-to-left implication:** Assume that  $\mathcal{I}nc_{R_1, R_2}$  is finitely colorable, say by  $(A_1, \dots, A_k)$ . Then let  $S$  be the union of all  $S_i$ 's where

$$\begin{aligned} S_i \doteq \{ & (u, v) \mid u \in A_i \text{ and } (u', v) \in R_1 \text{ for some } u' \in A_i \} \\ & \cup \{ (u, v) \mid v \in A_i \text{ and } (u, v') \in R_1 \text{ for some } v' \in A_i \}. \end{aligned}$$

Since  $(A_1, \dots, A_k)$  covers every node of  $\mathcal{I}nc_{R_1, R_2}$ , we get  $R_1 \subseteq S$ . Moreover, we claim that  $R_2 \cap S = \emptyset$ . Indeed, if  $(u, v) \in S$ , then  $(u, v) \in S_i$  for some  $i, j$ . It either means that **1**  $(u', v) \in R_1$  for some  $u' \in A_i$ , or **2**  $(u, v') \in R_1$  for some  $v' \in A_i$ . In case **1**, the fact that  $u \in A_i$  implies that  $u$  and  $u'$  have the same color. Thus,  $u$  must be compatible with  $u'$  and hence  $(u, v) \notin R_2$  using  $(\text{COMP}'_\ell)$ . The other case is symmetric. Therefore,  $(u, v) \notin R_2$ , and thus  $S$  separates  $R_1$  from  $R_2$ .

Finally,  $S$  is recognizable; in fact,  $S = \bigcup_{i=1}^k (A_i \times R_1[A_i]) \cup (R_1^{-1}[A_i] \times A_i)$ , where for any set  $X \subseteq \mathbb{A}^*$  we define  $R_1[X]$  (resp.  $R_1^{-1}[X]$ ) as the set of  $v \in \mathbb{A}^*$  (resp.  $u \in \mathbb{A}^*$ ) such that  $(u, v) \in R_1$  for some  $u \in X$  (resp.  $v \in X$ ). Hence,  $R_1$  and  $R_2$  are REC-separable. ◀

It is not known to date whether the REGULAR COLORABILITY PROBLEM is decidable, and hence the same holds for the REC-SEPARABILITY PROBLEM in light of the previous theorem. This is due to the fact that there are no known characterizations of when an automatic graph is finitely colorable. In spite of this, we believe that the connection between separability and finite colorability is of interest, as it provides us with a way to define and study meaningful restrictions of our problems. The first such restriction corresponds to the  $k$ -REGULAR COLORABILITY PROBLEM for automatic graphs, which we study in the next section.

## 4 $k$ -Regular Colorability Problem

While we do not know how to approach the REGULAR COLORABILITY PROBLEM, we show that as soon as we add the restriction that the number of colors is bounded, the problem becomes undecidable; i.e., the  $k$ -REGULAR COLORABILITY PROBLEM is undecidable for  $k \geq 2$ . Using this, we obtain in the next section the undecidability for the SEPARABILITY PROBLEM on two natural classes of recognizable relations. This is proven by a reduction from a suitable problem on reversible Turing Machines with certain restrictions, which we call “well-founded”.

### 4.1 Regularity of Reachability for Turing Machines

We use the standard notation  $u[i..j]$  to denote the factor of a word  $u$  between (and including) positions  $i$  and  $j$ , and  $u[i]$  to denote  $u[i..i]$ . Consider any deterministic Turing Machine (TM)  $T = \langle Q, \Gamma, \perp, \delta, q_0, F \rangle$ , where  $Q$  is the set of states,  $\Gamma$  is tape alphabet,  $\perp$  is the blank symbol,  $\delta : (Q \setminus F) \times \Gamma_{\perp} \rightarrow Q \times \Gamma \times \{L, R\}$  is the transition (partial) function, where  $\Gamma_{\perp} = \Gamma \cup \{\perp\}$ , and  $q_0$  and  $F$  is the initial and set of final states, respectively. We represent a configuration with tape content  $w \cdot \perp^{\omega}$  (where  $w \in \Gamma^* \cdot \{\perp\}$ ), in state  $q$  and with the head pointing to the cell number  $1 \leq i \leq |w|$ , as the string

$$w[1..i-1] \cdot (w[i], q) \cdot w[i+1..|w|]$$

over the alphabet  $\mathbb{A}_T = \Gamma \cup (\Gamma_{\perp} \times Q)$ . In light of this representation, we will henceforth denote by “configuration” any string from the set  $\text{Conf}_T \hat{=} (\Gamma^* \cdot (\Gamma_{\perp} \times Q)) \cup (\Gamma^* \cdot (\Gamma \times Q) \cdot \Gamma^*)$ . The *initial configuration* is  $(\perp, q_0)$ . The *configuration graph* of  $T$  is the infinite graph  $\mathcal{G}^T$  having  $\text{Conf}_T$  as set of vertices and an edge from  $c$  to  $c'$ , denoted  $c \rightarrow c'$ , if  $c'$  is the configuration of the next step of  $T$  starting from  $c$ . Observe that the configuration graph  $\mathcal{G}^T$  of any TM  $T$  is an effective automatic graph (see, e.g., [18]).

We say that a deterministic TM  $T$  is *reversible* if every node of  $\mathcal{G}^T$  has in-degree at most 1, in other words if the machine is co-deterministic<sup>4</sup>. We say that a TM  $T$  is a *well-founded Reversible Turing Machine (wf-RTM)* if its configuration graph is such that (1) the initial configuration has in-degree 0 (2) every node has in-degree and out-degree at most one (3) there are no infinite backward paths  $c_1 \leftarrow c_2 \leftarrow \dots$  in  $\mathcal{G}^T$ .

Note that every well-founded Reversible Turing Machine is deterministic and reversible and, moreover, its configuration graph is a (possibly infinite) disjoint union of directed paths, which are all finite, or isomorphic to  $(\mathbb{N}, +1)$ . The set of *reachable configurations*, denoted by  $\text{Reach}$ , is the set of all configurations that admit a path from the initial configuration in  $\mathcal{G}^T$ , for a given TM  $T$ . Such a configuration graph is depicted on Figure 2a.

The *REACHABLE REGULARITY PROBLEM* is the problem of, given a wf-RTM  $T$ , whether its set of reachable configurations is a regular language. To show that it is undecidable, we exhibit a reduction from the halting problem on deterministic reversible Turing machines.

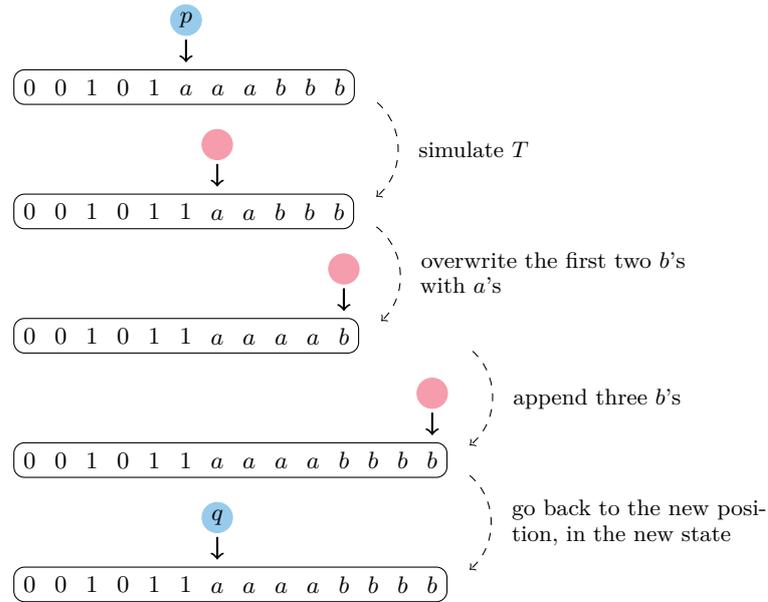
► **Proposition 4.1** ([19, Theorem 1]). *The halting problem on deterministic reversible Turing machines is undecidable.*

For more details and pointers on reversible Turing machines, see [22, Chapter 5].

► **Lemma 4.2.** *The REACHABLE REGULARITY PROBLEM is undecidable.*

<sup>4</sup> Note that a modern proof of undecidability of the isomorphism problem for automatic structures by Blumensath [6, §VIII. Theorem 4.3, p. 396 & second claim, p. 398] also relies on the use of reversible Turing machines.

**Proof sketch.** By reducing the halting problem on deterministic reversible Turing machines, in such a way that the reachable configurations whose state  $q$  coincide with the state of the original machine are of the form  $(uqva^n b^n)$  where  $(uqv)$  is a configuration of the original machine,  $a$  and  $b$  are new symbols, and  $n \in \mathbb{N}$ . Transitions are defined in such a way that the new machine is a wf-RTM: this is implemented by having, for every transition  $uqv \rightarrow u'q'v'$  of the original machine and every  $n \in \mathbb{N}$ , a (multi-step) transition  $(uqva^n b^n) \rightarrow^* (u'q'v'a^{n+1}b^{n+1})$  – and is illustrated in Figure 1. Moreover:



■ **Figure 1** Encoding of a single transition of the form “when reading a blank in state  $p$ , write a 1, go in state  $q$  and move right” of the machine  $T$  in the machine  $T'$  in the proof of Lemma 4.2. Red unlabelled states represent states of  $T'$  that are not originally present in  $T$ .

- if the original machine was halting, then the reachable configurations of the new one are finite and hence regular;
- otherwise, the set of reachable configurations is not regular, which follows from the non-regularity of any infinite subset of  $\{a^n b^n \mid n \in \mathbb{N}\}$ .

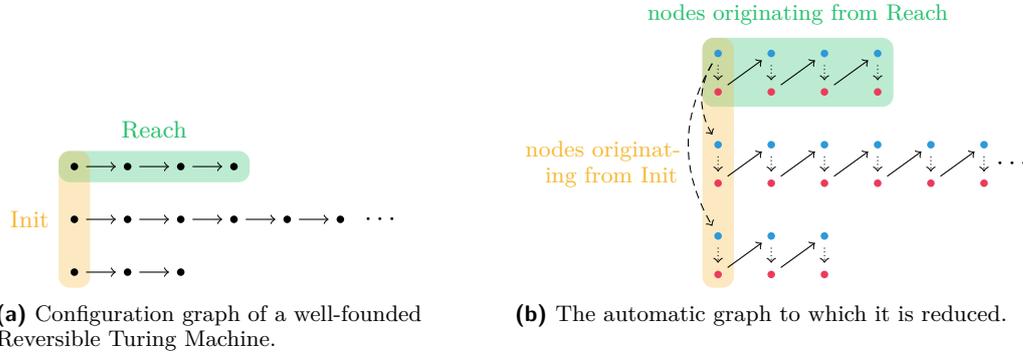
See the full version for more details. ◀

## 4.2 Undecidability of the $k$ -Regular Colorability Problem

We can now show undecidability for the  $k$ -REGULAR COLORABILITY PROBLEM by reduction from the REACHABLE REGULARITY PROBLEM as defined before.

▷ **Fact 4.3.** Given an automatic graph, the set of nodes with no predecessor is effectively a regular language.

► **Theorem 4.4.** *The  $k$ -REGULAR COLORABILITY PROBLEM on automatic graphs is undecidable, for every  $k \geq 2$ . More precisely, the problem is recursively enumerable-complete. This holds also for connected automatic graphs.*



■ **Figure 2** Reduction used in the proof of Theorem 4.4.

**Proof.** **Lower bound.** By reduction from the REACHABLE REGULARITY PROBLEM for wf-RTMs (Lemma 4.2). We first show it for  $k = 2$ . Given a wf-RTM  $T$ , let  $c_{init}$  be its initial configuration. Observe that the set  $Init$  of all vertices of  $\mathcal{G}^T$  with in-degree 0 is an effective regular language (by Fact 4.3), and that  $c_{init} \in Init$ . Let  $B$  and  $R$  be fresh symbols. Consider the automatic graph  $\langle L, E \rangle$  for  $L = \{B, R\} \times Confs_T$ , having an edge from  $(z, c) \in \{B, R\} \times Confs_T$  to  $(z', c') \in \{B, R\} \times Confs_T$  if either

1.  $(z, z') = (B, R)$  and  $c = c'$ ;
2.  $(z, z') = (R, B)$  and there is an edge from  $c$  to  $c'$  in  $\mathcal{G}^T$ ; or
3.  $(z, z') = (B, B)$ ,  $c = c_{init}$  and  $c' \in Init \setminus \{c_{init}\}$ .

Fresh symbols  $B$  and  $R$  are utilized to represent two versions of each configuration - one in Blue and one in Red. This graph is depicted on Figure 2. Note that  $\langle L, E \rangle$  is connected and 2-colorable: in fact, it is a directed (possibly infinite) tree with root  $(B, c_{init})$ .

We claim that  $\langle L, E \rangle$  is 2-regular colorable if, and only if, the set of reachable configurations of  $T$  is a regular language. In fact, up to permuting the two-colors,  $\langle L, E \rangle$  admits a unique 2-coloring, defined by:

$$C_1 \hat{=} \{B\} \times Reach \cup \{R\} \times (Confs_T \setminus Reach)$$

and  $C_2$  is the complement of  $C_1$ . If  $Reach$  is regular, then so is  $C_1$ . Dually, if  $C_1$  is regular, then  $Reach$  is the set of configurations  $c$  such that  $(B, c) \in C_1$  and hence is regular. It follows that  $\langle \mathbb{A}^*, E \rangle$  is 2-regular colorable if and only if the reachable configurations of  $T$  are regular, which concludes the proof for  $k = 2$ .

To prove the statement for any  $k > 2$ , we define  $\langle L, E_k \rangle$  as the result of adding a  $(k - 2)$ -clique to  $\langle L, E \rangle$  and adding an edge from every vertex of the clique to every vertex incident to an edge of  $E$ . This forces the clique to use  $k - 2$  colors that cannot be used in the remaining part of the graph and the proof is then analogous.

**Upper-bound.** We show that the problem is recursively enumerable. Let us define a  $k$ -colored automaton like a regular (complete) DFA, except that instead of having a set of final states, it has a partition  $\langle C_1, \dots, C_k \rangle$  of its states. Such an automaton recognizes a regular coloring  $\mathbb{A}^* \rightarrow \{1, \dots, k\}$ . Given an automatic graph  $\langle L, R \rangle$  – specified by NFA’s  $\mathcal{A}_1$  and  $\mathcal{A}_2$  recognizing  $L$  and  $\otimes R$  respectively – and a  $k$ -colored automaton  $\mathcal{B}$ , we can build, by a product construction, an NFA  $\mathcal{A}'_2$  which accepts all  $u \otimes v \in \otimes R$  such that the color of  $u$  is

## 17:10 Separating Automatic Relations

distinct from the color of  $v$ . Then,  $\mathcal{A}'_2$  is equivalent to  $\mathcal{A}_2$  if, and only if,  $\mathcal{B}$  describes a proper  $k$ -coloring of  $\langle L, R \rangle$ . The RE upper-bound of the  $k$ -REGULAR COLORABILITY PROBLEM follows: it suffices to enumerate all  $k$ -colored automata and check for equivalence. ◀

Note that this reduction provides an easy way of building graphs in the shape of Figure 2b that are 2-colorable (in fact, they are trees) but not 2-regular colorable. In fact, we can provide a slightly more direct construction.

► **Example 4.5.** On the alphabet  $\mathbb{A} = \{a, b\}$ , the tree  $\mathcal{T}$  depicted in Figure 3 whose set of vertices is  $V = a^*b^*$  and whose set of edges is  $E = E_{\text{incr}} \cup E_{\text{init}}$ , with

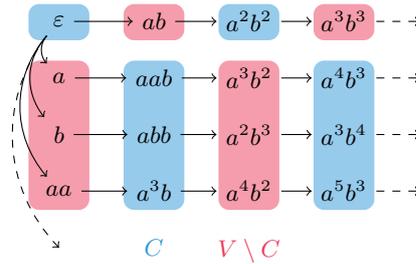
$$E_{\text{incr}} = \{(a^p b^q, a^{p+1} b^{q+1}) \mid p, q \in \mathbb{N}\}$$

$$E_{\text{init}} = \{(\varepsilon, a^p) \mid p \in \mathbb{N}\} \cup \{(\varepsilon, b^q) \mid q \in \mathbb{N}\},$$

is automatic but not 2-regular colorable. Indeed, its only 2-coloring consists in partitioning the vertices of  $\mathcal{T}$  into

$$C = \{a^n b^n \mid n \in 2\mathbb{N}\} \cup \{a^p b^q \mid p > q \text{ and } q \text{ is odd}\} \cup \{a^p b^q \mid p < q \text{ and } p \text{ is odd}\}$$

and its complement  $V \setminus C$ . Let  $P = \{a^p b^q \mid p, q \in 2\mathbb{N}\} = (aa)^*(bb)^*$ :  $P$  is regular, yet  $C \cap P = \{a^n b^n \mid n \in 2\mathbb{N}\}$  is not. Hence,  $C$  is not regular, and thus  $\mathcal{T}$  is not 2-regular colorable. ◻



■ **Figure 3** The automatic tree  $\mathcal{T}$  of Example 4.5, and its unique 2-coloring  $(C, V \setminus C)$ , which is not regular.

## 5 Separability for Bounded Recognizable Relations

In this section we capitalize on the undecidability result of the previous section, showing how this implies the undecidability for the SEPARABILITY PROBLEM on two natural classes of bounded recognizable relations, namely:  $k$ -REC, and  $k$ -PROD. Remember that, for any  $k$ ,  $k$ -PROD is the subclass of REC consisting of unions of  $k$  cross-products of regular languages (which is a subclass of  $2^{2k}$ -REC).

**$k$ -Rec-separability.** First, observe that the 1-REC-SEPARABILITY PROBLEM is trivially decidable, since the only possible separator is  $\mathbb{A}^* \times \mathbb{A}^*$ . However, for any other  $k > 1$ , the problem is undecidable.

► **Proposition 5.1.** *The  $k$ -REC-SEPARABILITY PROBLEM is undecidable, for every  $k > 1$ .*

**Proof.** A consequence of the reduction from the  $k$ -REGULAR COLORABILITY PROBLEM of Theorem 3.1, combined with the undecidability of the latter for every  $k > 1$  (Theorem 4.4). ◀

**$k$ -PROD-separability.** On the  $k$ -PROD hierarchy we will find the same phenomenon. In particular the case  $k = 1$  is also trivially decidable.

► **Proposition 5.2.** *The 1-PROD-SEPARABILITY PROBLEM is decidable.*

**Proof.** Given two automatic relations  $R_1, R_2$ , there exists  $S \in 1\text{-PROD}$  that separates  $R_1$  from  $R_2$  if and only if  $\pi_1(R_1) \times \pi_2(R_1)$  separates  $R_1$  from  $R_2$ . ◀

As soon as  $k > 1$ , the  $k$ -PROD SEPARABILITY PROBLEM becomes undecidable. This is a consequence of the following simple lemma.

► **Lemma 5.3.** *A symmetric automatic relation  $R$  and the identity  $Id$  are separable by a relation in 2-PROD iff they have a separator of the form  $(A \times B) \cup (B \times A)$ .*

**Proof.** Assume that  $S \in 2\text{-PROD}$  separates  $R$  from  $Id$ . Then  $R \subseteq S$ , but since  $R$  is symmetric,  $R = R^{-1} \subseteq S^{-1}$  so  $R \subseteq S \cap S^{-1}$ , and hence  $R \subseteq S \cap S^{-1}$ . Moreover, since  $S$  has a trivial intersection with  $Id$ , so does  $S \cap S^{-1}$ . Hence,  $S \cap S^{-1}$  separates  $R$  from  $Id$ .

Since  $S \in 2\text{-PROD}$ , there exists  $A_1, A_2, B_1, B_2 \subseteq \mathbb{A}^*$  such that  $S = A_1 \times B_1 \cup B_2 \times A_2$ . Note that  $S \cap Id = \emptyset$  yields  $A_i \cap B_i = \emptyset$  for each  $i \in \{1, 2\}$ . Finally:

$$\begin{aligned}
 S \cap S^{-1} &= (A_1 \times B_1 \cup B_2 \times A_2) \cap (B_1 \times A_1 \cup A_2 \times B_2) \\
 &= ((A_1 \times B_1) \cap (B_1 \times A_1)) \cup ((A_1 \times B_1) \cap (A_2 \times B_2)) \\
 &\quad \cup ((B_2 \times A_2) \cap (B_1 \times A_1)) \cup ((B_2 \times A_2) \cap (A_2 \times B_2)) \\
 &= \overbrace{((A_1 \cap B_1) \times (A_1 \cap B_1))}^{=\emptyset} \cup ((A_1 \cap A_2) \times (B_1 \cap B_2)) \\
 &\quad \cup ((B_1 \cap B_2) \times (A_1 \cap A_2)) \cup \underbrace{((A_2 \cap B_2) \times (A_2 \cap B_2))}_{=\emptyset} \\
 &= ((A_1 \cap A_2) \times (B_1 \cap B_2)) \cup ((B_1 \cap B_2) \times (A_1 \cap A_2)). \quad \blacktriangleleft
 \end{aligned}$$

We can then establish the following:

► **Corollary 5.4.** *A symmetric automatic relation  $R$  and  $Id$  are separable by a relation in 2-PROD iff  $\langle \mathbb{A}^*, R \rangle$  is 2-regular colorable.*

**Proof.** By observing that for any symmetric relation  $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ , we have that  $A, B \subseteq \mathbb{A}^*$  is a coloring of  $\langle \mathbb{A}^*, R \rangle$  if, and only if,  $(A \times B) \cup (B \times A)$  separates  $R$  from  $Id$ . ◀

We can now easily show undecidability for the 2-PROD SEPARABILITY PROBLEM by reduction from the 2-REGULAR COLORABILITY PROBLEM.

► **Lemma 5.5.** *The 2-PROD-SEPARABILITY PROBLEM is undecidable.*

**Proof.** By reduction from the 2-REGULAR COLORABILITY PROBLEM on automatic graphs, which is undecidable by Theorem 4.4. Let  $\langle L, R \rangle$  be an automatic graph and  $\langle L, R' \rangle$  the symmetric closure of  $\langle L, R \rangle$ . It follows that  $\langle L, R' \rangle$  is still automatic and that there is a 2-regular coloring for  $\langle L, R' \rangle$  iff there is a 2-regular coloring for  $\langle L, R \rangle$  (the same coloring in fact). Thus, by Corollary 5.4,  $\langle L, R \rangle$  is 2-regular colorable iff there is a 2-PROD relation that separates  $R'$  from  $Id$ . ◀

Further, this implies undecidability for every larger  $k$ :

► **Theorem 5.6.** *The  $k$ -PROD-SEPARABILITY PROBLEM is undecidable, for every  $k \geq 2$ .*



1.  $(w, v) \in R$  iff  $(w', v) \in R$ , and
2.  $(v, w) \in R$  iff  $(v, w') \in R$ .

It turns out that equivalence classes of  $\sim_R$  define the coarsest partition onto which  $R$  can be recognized in terms of  $k$ -REC:

► **Lemma 6.1.** *For every automatic  $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ ,  $\sim_R$  has index at most  $k$  if, and only if,  $R$  is in  $k$ -REC.*

**Proof.** **Left-to-right** Assume that  $\sim_R$  has the equivalence classes  $E_1, \dots, E_k$ . Consider the set  $P \subseteq \{1, \dots, k\}^2$  of all pairs  $(i, j)$  such that there are  $u_i \in E_i$  and  $u_j \in E_j$  with  $(u_i, u_j) \in R$ . Define the  $k$ -REC relation  $R' = \bigcup_{(i,j) \in P} E_i \times E_j$ . We claim that  $R = R'$ . In fact, by definition of  $\sim_R$ , note that if there are  $u_i \in E_i$  and  $u_j \in E_j$  with  $(u_i, u_j) \in R$ , then  $E_i \times E_j \subseteq R$ . Hence,  $R' \subseteq R$ . On the other hand, for every pair  $(u, v) \in R$  there is  $(i, j) \in P$  such that  $u \in E_i, v \in E_j$  implying  $(u, v) \in R'$ . Hence,  $R \subseteq R'$ .

**Right-to-left** If  $R$  is a union of products of sets from the partition  $E_1 \dot{\cup} \dots \dot{\cup} E_k = \mathbb{A}^*$ , then every two elements of each  $E_i$  are  $\sim_R$ -related, and thus  $\sim_R$  has index at most  $k$ . ◀

We can then conclude that the definability problem for  $k$ -REC is decidable.

► **Corollary 6.2.** *The  $k$ -REC-DEFINABILITY PROBLEM is decidable, for every  $k > 0$ .*

**Proof.** An automatic relation  $R$  is in  $k$ -REC iff  $\sim_R$  has at most  $k$  equivalence classes by Lemma 6.1. In other words, an automatic relation  $R$  is not in  $k$ -REC iff the complement of  $\sim_R$  contains a  $(k+1)$ -clique, which can be easily tested. ◀

The relation  $\sim_R$  can also be used to characterize which automatic relations are definable in the class  $k$ -PROD.

► **Lemma 6.3.** *An automatic relation  $R$  is in  $k$ -PROD if, and only if,  $R = (A_1 \times B_1) \cup \dots \cup (A_k \times B_k)$  where each  $A_i$  and  $B_i$  is a union of equivalence classes of  $\sim_R$ .*

**Proof.** It suffices to show that for every equivalence class  $E$  from  $\sim_R$ , if  $A_1 \cap E \neq \emptyset$  then  $R = ((A_1 \cup E) \times B_1) \cup \dots \cup (A_k \times B_k)$ , and similarly for  $B_1$ . Assume  $w \in A_1 \cap E$  and take any pair  $(u, v) \in E \times B_1$ . We show that  $(u, v) \in R$ . By definition of  $\sim_R$ , since  $(w, v) \in R$  and  $w \sim_R u$ , we have that  $(u, v) \in R$ . ◀

Again, this characterization allows us to show that definability in the class  $k$ -PROD is decidable.

► **Corollary 6.4.** *The  $k$ -PROD-DEFINABILITY PROBLEM is decidable, for every  $k > 0$ .*

**Proof.** By brute force testing whether the automatic relation  $R$  is equivalent to  $(A_1 \times B_1) \cup \dots \cup (A_k \times B_k)$  for every possible  $A_i, B_i$  which is a union of equivalence classes of  $\sim_R$ . ◀

## 7 Discussion

We have established, among other things, the undecidability of the  $k$ -REGULAR COLORABILITY PROBLEM for  $k \geq 2$ . Yet, little is known about the REGULAR COLORABILITY PROBLEM.

► **Conjecture 7.1.** *The REC-SEPARABILITY PROBLEM – or, equivalently, the REGULAR COLORABILITY PROBLEM – is undecidable.*

Beyond its decidability status, the structural properties of regular colorability evades us:

► **Conjecture 7.2.** *Over automatic graphs, the following notions are pairwise disjoint:*

1. *to be finitely regular colorable,*
2. *to be finitely colorable,*
3. *not to contain unbounded cliques.*

Note that the implications (1)  $\Rightarrow$  (2)  $\Rightarrow$  (3) trivially hold. Moreover, recall that while the automatic tree of Example 4.5 is not 2-regular colorable, it is 3-regular colorable (it suffices to color  $\varepsilon$  with a new color, and then color  $a^p b^q$  by looking at the parity of  $p - q$ ). Hence, it does not prove that (2)  $\not\Rightarrow$  (1). Likewise, on arbitrary infinite graphs, we know that there exists triangle-free graphs that are not finitely colorable [26] – but we believe these graphs not to be automatic, and hence they would not prove that (3)  $\not\Rightarrow$  (2).

Finally, observe that it is decidable to test whether an automatic graph has *infinite* cliques [18, Corollary 5.5]. We conjecture that this property generalizes to unbounded cliques.

► **Conjecture 7.3.** *The problem of whether an automatic graph has bounded cliques is decidable.*

---

## References

- 1 Pablo Barceló, Chih-Duo Hong, Xuan Bach Le, Anthony W. Lin, and Reino Niskanen. Monadic decomposability of regular relations. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 103:1–103:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.103.
- 2 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):31, 2012. doi:10.1145/2389241.2389250.
- 3 Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *Journal of the ACM*, 50(5):694–751, 2003. doi:10.1145/876638.876642.
- 4 Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetsche. Ramsey quantifiers over automatic structures: Complexity and applications to verification. In Christel Baier and Dana Fisman, editors, *Annual Symposium on Logic in Computer Science (LICS)*, pages 28:1–28:14. ACM, 2022. doi:10.1145/3531130.3533346.
- 5 Jean Berstel. *Transductions and Context-Free Languages*. Teubner-Verlag, 1979.
- 6 Achim Blumensath. Monadic Second-Order Model Theory. Preprint of a book., 2023. Version of 2023-02-25. URL: <https://www.fi.muni.cz/~blumens/MSO.pdf>.
- 7 Achim Blumensath and Erich Grädel. Automatic structures. In *Annual Symposium on Logic in Computer Science (LICS)*, pages 51–62. IEEE Computer Society, 2000.
- 8 Olivier Carton, Christian Choffrut, and Serge Grigorieff. Decision problems among the main subfamilies of rational relations. *RAIRO – Theoretical Informatics and Applications*, 40(2):255–275, 2006. doi:10.1051/ita:2006005.
- 9 Christian Choffrut. Relations over words and logic: A chronology. *Bull. of the EATCS*, 89:159–163, 2006.
- 10 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular separability of Parikh automata. In *ICALP*, pages 117:1–117:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.117.
- 11 Wojciech Czerwiński, Wim Martens, Larijn van Rooijen, Marc Zeitoun, and Georg Zetsche. A characterization for decidable separability by piecewise testable languages. *Discret. Math. Theor. Comput. Sci.*, 19(4), 2017.
- 12 Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.
- 13 Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993. doi:10.1016/0304-3975(93)90230-q.

- 14 Hajime Ishihara, Bakhadyr Khossainov, and Sasha Rubin. Some results on automatic structures. In *Annual Symposium on Logic in Computer Science (LICS)*, page 235. IEEE Computer Society, 2002. doi:10.1007/11690634\_22.
- 15 Chris Köcher. *Analyse der Entscheidbarkeit diverser Probleme in automatischen Graphen*. Unpublished manuscript, 2014. URL: <https://people.mpi-sws.org/~ckoecher/files/theses/bsc-thesis.pdf>.
- 16 Eryk Kopczyński. Invisible pushdown languages. In *Annual Symposium on Logic in Computer Science (LICS)*, pages 867–872, 2016. doi:10.1145/2933575.2933579.
- 17 Dietrich Kuske and Markus Lohrey. Hamiltonicity of automatic graphs. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *IFIP*, volume 273, pages 445–459. Springer, 2008.
- 18 Dietrich Kuske and Markus Lohrey. Some natural decision problems in automatic graphs. *J. Symb. Log.*, 75(2):678–710, 2010. doi:10.2178/js1/1268917499.
- 19 Yves Lecerf. Machines de turing réversibles. récursive insolubilité en  $n \in \mathbb{N}$  de l'équation  $u = \theta^n u$ , ou  $\theta$  est un «isomorphisme de codes». *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 257:2597–2600, 1963.
- 20 Anthony W. Lin and Pablo Barceló. String solving with word equations and transducers: Towards a logic for analysing mutation XSS. In *Annual Symposium on Principles of Programming Languages (POPL)*, pages 123–136. ACM, 2016. doi:10.1145/2837614.2837641.
- 21 Christof Löding and Christopher Spinrath. Decision problems for subclasses of rational relations over finite and infinite words. *Discret. Math. Theor. Comput. Sci.*, 21(3), 2019. doi:10.23638/DMTCS-21-3-4.
- 22 Kenichi Morita. *Reversible Turing Machines*, pages 103–156. Springer Japan, Tokyo, 2017. doi:10.1007/978-4-431-56606-9\_5.
- 23 Maurice Nivat. Transduction des langages de Chomsky. *Ann. Inst. Fourier*, 18:339–455, 1968.
- 24 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science (LMCS)*, 12(1), 2016. doi:10.2168/LMCS-12(1:5)2016.
- 25 Richard Edwin Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967. doi:10.1016/S0019-9958(67)90591-8.
- 26 Peter Ungar and Blanche Descartes.  $k$ -Chromatic graphs without triangles. *The American Mathematical Monthly*, 61(5):352–353, 1954. doi:10.2307/2307489.
- 27 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10, 1975. doi:10.1145/321864.321865.



# On the Parameterized Complexity of Computing $st$ -Orientations with Few Transitive Edges

Carla Binucci  

Department of Engineering, University of Perugia, Italy

Giuseppe Liotta  

Department of Engineering, University of Perugia, Italy

Fabrizio Montecchiani  

Department of Engineering, University of Perugia, Italy

Giacomo Ortali  

Department of Engineering, University of Perugia, Italy

Tommaso Piselli  

Department of Engineering, University of Perugia, Italy

---

## Abstract

Orienting the edges of an undirected graph such that the resulting digraph satisfies some given constraints is a classical problem in graph theory, with multiple algorithmic applications. In particular, an  $st$ -orientation orients each edge of the input graph such that the resulting digraph is acyclic, and it contains a single source  $s$  and a single sink  $t$ . Computing an  $st$ -orientation of a graph can be done efficiently, and it finds notable applications in graph algorithms and in particular in graph drawing. On the other hand, finding an  $st$ -orientation with at most  $k$  transitive edges is more challenging and it was recently proven to be NP-hard already when  $k = 0$ . We strengthen this result by showing that the problem remains NP-hard even for graphs of bounded diameter, and for graphs of bounded vertex degree. These computational lower bounds naturally raise the question about which structural parameters can lead to tractable parameterizations of the problem. Our main result is a fixed-parameter tractable algorithm parameterized by treewidth.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Fixed parameter tractability; Mathematics of computing  $\rightarrow$  Graph algorithms

**Keywords and phrases**  $st$ -orientations, parameterized complexity, graph drawing

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.18

**Related Version** *Full Version*: <http://arxiv.org/abs/2306.03196> [2]

**Funding** Research partially supported by (i) University of Perugia, Fondi di Ricerca di Ateneo, edizione 2021, project “AIDMIX - Artificial Intelligence for Decision making: Methods for Interpretability and eXplainability”, (ii) Progetto RICBA22CB “*Modelli, algoritmi e sistemi per la visualizzazione e l’analisi di grafi e reti*”.

## 1 Introduction

An orientation of an undirected graph is an assignment of a direction to each edge, turning the initial graph into a directed graph (or *digraph* for short). Notable examples of orientations are acyclic orientations, which guarantee the resulting digraph to be acyclic; transitive orientations, which make the resulting digraph its own transitive closure; and Eulerian orientations, in which each vertex has equal in-degree and out-degree. Of particular interest for our research are certain constrained acyclic orientations, which find applications in several domains, including graph planarity and graph drawing. More specifically, given a graph  $G = (V, E)$  and two vertices  $s, t \in V$ , an  $st$ -orientation of  $G$ , also known as *bipolar*



© Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, Giacomo Ortali, and Tommaso Piselli; licensed under Creative Commons License CC-BY 4.0

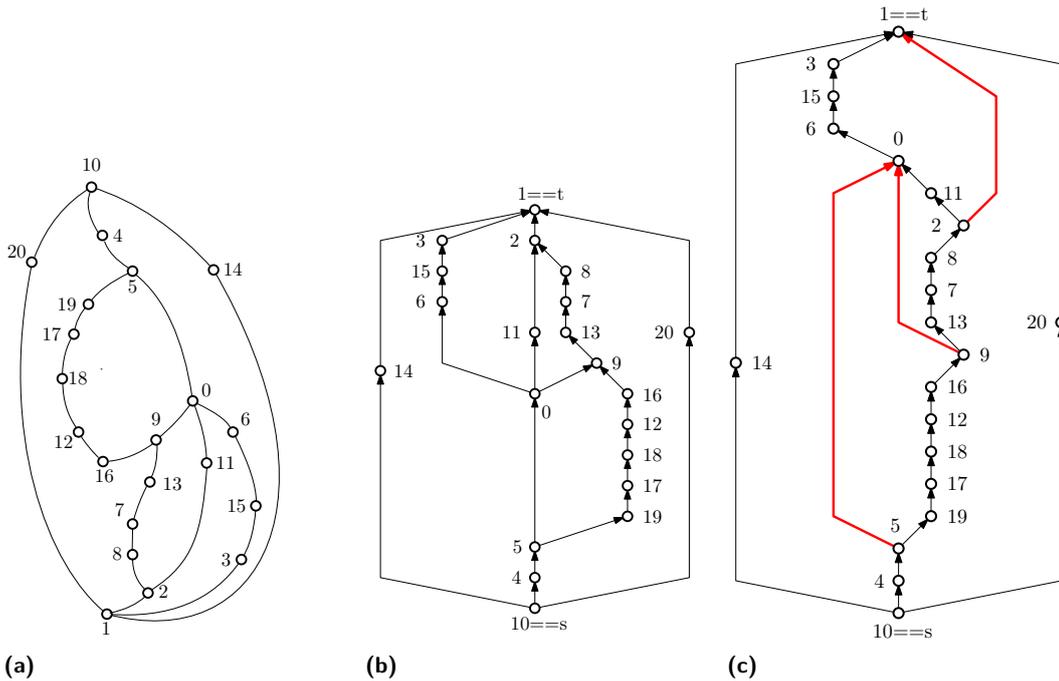
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 18; pp. 18:1–18:15

Leibniz International Proceedings in Informatics



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

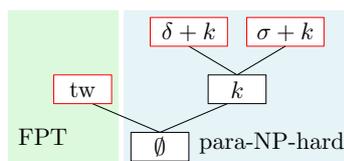


■ **Figure 1** (a): An undirected graph  $G$  with randomly labeled vertices. (b)-(c): Two polyline drawings of  $G$  computed by using different  $st$ -orientations. The drawing in (b) uses an  $st$ -orientation without transitive edges and it has smaller area and number of bends than the drawing in (c).

*orientation*, is an orientation of its edges such that the corresponding digraph is acyclic and contains a single source  $s$  and a single sink  $t$ . It is well-known that  $G$  admits an  $st$ -orientation if and only if it is biconnected after the addition of the edge  $st$  (if not already present). The computation of an  $st$ -numbering (an equivalent concept defined on the vertices of the graph) is for instance part of the quadratic-time planarity testing algorithm by Lempel, Even and Cederbaum [11]. Later, Even and Tarjan [8] showed how to compute an  $st$ -numbering in linear time, and used this result to derive a linear-time planarity testing algorithm. In the field of graph drawing, bipolar orientations are a central algorithmic tool to compute different types of layouts, including visibility representations, polyline drawings, dominance drawings, and orthogonal drawings (see [5, 9] for references). On a similar note, a notable result states that a planar digraph admits an upward planar drawing if and only if it is the subgraph of a planar  $st$ -graph, that is, a planar digraph with a bipolar orientation [6].

Recently, Binucci, Didimo and Patrignani [1] focused on  $st$ -orientations with no transitive edges. We recall that an edge  $uv$  is *transitive* if the digraph contains a path directed from  $u$  to  $v$ ; for example, the bold (red) edges in Figure 1c are transitive, see also Section 2 for formal definitions. Besides being of theoretical interest, such orientations, when they exist, can be used to compute readable and compact drawings of graphs [1]. For example, a classical graph drawing algorithm relies on  $st$ -orientations to compute polyline representations of planar graphs. The algorithm is such that both the height and the number of bends of the representations can be reduced by computing  $st$ -orientations with few transitive edges. See Algorithm `PolyLine` in [5] for details and Figure 1 for an example.

Unfortunately, while an  $st$ -orientation of an  $n$ -vertex graph can be computed in  $O(n)$  time, computing one that has the minimum number of transitive edges is much more challenging from a computational perspective. Namely, Binucci et al. [1] prove that the problem of deciding whether an  $st$ -orientation with no transitive edges exists is NP-complete, and provide an ILP model for planar graphs.



■ **Figure 2** The complexity landscape of the ST-ORIENTATION problem. The symbols  $tw$ ,  $\delta$ , and  $\sigma$  denote the treewidth, the maximum vertex degree, and the diameter of the graph, respectively. The boxes with red boundaries denote the new results presented in this paper.

**Contribution.** We study the parameterized complexity of finding  $st$ -orientations with few transitive edges. More formally, given a graph  $G$  and an integer  $k$ , the ST-ORIENTATION problem asks for an  $st$ -orientation of  $G$  with at most  $k$  transitive edges (see also Section 2). As already discussed, ST-ORIENTATION is para-NP-hard by the natural parameter  $k$  [1]. We strengthen this result by showing that, for  $k = 0$ , ST-ORIENTATION remains NP-hard even for graphs of diameter at most six, and for graphs of vertex degree at most four. In light of these computational lower bounds, we seek for structural parameters that can lead to tractable parameterizations of the problem. Our main result is a fixed-parameter tractable algorithm for ST-ORIENTATION parameterized by treewidth, a central parameter in the parameterized complexity analysis (see [7, 12]). Figure 2 depicts a summary of the computational complexity results known for the ST-ORIENTATION problem.

It is worth remarking that by Courcelle’s theorem one can derive an (implicit) FPT algorithm parameterized by treewidth and  $k$ , while we provide an explicit algorithm parameterized by treewidth only. The main challenge in applying dynamic programming over a tree decomposition is that the algorithm must know if adding an edge to the graph may cause previously forgotten non-transitive edges to become transitive, and, if so, how many of them. To tackle this difficulty, we describe an approach that avoids storing information about all edges that may potentially become transitive; instead, the algorithm guesses the edges that will be transitive in a candidate solution and ensures that no other edge will become transitive in the course of the algorithm. Our technique can be easily adapted to handle more general constraints on the sought orientation, for instance the presence of multiple sources and sinks.

**Paper structure.** We begin with preliminary definitions and basic tools, which can be found in Section 2. In Section 3 we describe our main result, an FPT algorithm for the ST-ORIENTATION problem parameterized by treewidth. Section 4 contains our second contribution, namely we adapt the NP-hardness proof in [1] to prove that the result holds also for graphs that have bounded diameter and for graphs with bounded vertex degree. In the latter case, the graphs used in the reduction not only have bounded vertex degree (at most four), but are also subdivisions of triconnected graphs. In Section 5 we list some interesting open problems that stem from our research.

For space reasons, some proofs have been omitted, and the corresponding statements are marked with  $(\star)$ .

## 2 Preliminaries

**Edge orientations.** Let  $G = (V, E)$  be an undirected graph. An *orientation*  $O$  of  $G$  is an assignment of a *direction*, also called *orientation*, to each edge of  $G$ . We denote by  $D_O(G)$  the digraph obtained from  $G$  by applying the orientation  $O$ . For each undirected pair  $(u, v) \in E$ , we write  $uv$  if  $(u, v)$  is oriented from  $u$  to  $v$  in  $D_O(G)$ , and we write  $vu$  otherwise. A directed

## 18:4 On the Parameterized Complexity of Computing $st$ -Orientations

path from a vertex  $u$  to a vertex  $v$  is denoted by  $u \rightsquigarrow v$ . A vertex of  $D_O(G)$  is a *source* (*sink*) if all its edges are outgoing (incoming). An edge  $uv$  of  $D_O(G)$  is *transitive* if  $D_O(G)$  contains a directed path  $u \rightsquigarrow v$  distinct from the edge  $uv$ . A digraph  $D_O(G)$  is an  *$st$ -graph* if: (i) it contains a single source  $s$  and a single sink  $t$ , and (ii) it is acyclic. An orientation  $O$  such that  $D_O(G)$  is an  *$st$ -graph* is called an  *$st$ -orientation*.

ST-ORIENTATION

**Input:** An undirected graph  $G = (V, E)$ , two vertices  $s, t \in V$ , and an integer  $k \geq 1$ .

**Output:** An  *$st$ -orientation*  $O$  of  $G$  such that the resulting digraph  $D_O(G)$  contains at most  $k$  transitive edges.

We recall that ST-ORIENTATION is NP-complete already for  $k = 0$  [1], which hinders tractability in the parameter  $k$ . Also, in what follows, we always assume that the input graph  $G$  is connected, otherwise we can immediately reject the instance as any orientation would give rise to at least one source and one sink for each connected component of  $G$ .

**Tree-decompositions.** Let  $(\mathcal{X}, T)$  be a pair such that  $\mathcal{X} = \{X_i\}_{i \in [\ell]}$  is a collection of subsets of vertices of a graph  $G = (V, E)$ , called *bags*, and  $T$  is a tree whose nodes are in one-to-one correspondence with the elements of  $\mathcal{X}$ . When this creates no ambiguity,  $X_i$  will denote both a bag of  $\mathcal{X}$  and the node of  $T$  whose corresponding bag is  $X_i$ . The pair  $(\mathcal{X}, T)$  is a *tree-decomposition* of  $G$  if:

1.  $\bigcup_{i \in [\ell]} X_i = V$ ,
2. For every edge  $uv$  of  $G$ , there exists a bag  $X_i$  that contains both  $u$  and  $v$ , and
3. For every vertex  $v$  of  $G$ , the set of nodes of  $T$  whose bags contain  $v$  induces a non-empty (connected) subtree of  $T$ .

The *width* of  $(\mathcal{X}, T)$  is  $\max_{i=1}^{\ell} |X_i| - 1$ , while the *treewidth* of  $G$ , denoted by  $\text{tw}(G)$ , is the minimum width over all tree-decompositions of  $G$ . The problem of computing a tree-decomposition of width  $\text{tw}(G)$  is fixed-parameter tractable in  $\text{tw}(G)$  [3]. A tree-decomposition  $(\mathcal{X}, T)$  of a graph  $G$  is *nice* if  $T$  is a rooted binary tree with the following additional properties [4]:

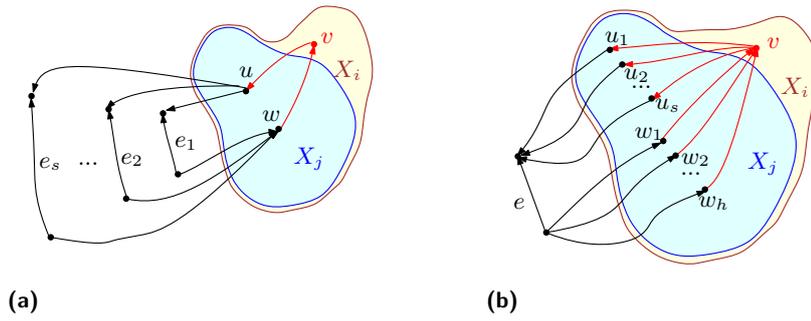
1. If a node  $X_i$  of  $T$  has two children whose bags are  $X_j$  and  $X_{j'}$ , then  $X_i = X_j = X_{j'}$ . In this case,  $X_i$  is a *join bag*.
2. If a node  $X_i$  of  $T$  has only one child  $X_j$ , then  $X_i \neq X_j$  and there exists a vertex  $v \in G$  such that either  $X_i = X_j \cup \{v\}$  or  $X_i \cup \{v\} = X_j$ . In the former case  $X_i$  is an *introduce bag*, while in the latter case  $X_i$  is a *forget bag*.
3. If a node  $X_i$  is the root or a leaf of  $T$ , then  $X_i = \emptyset$ . In this case,  $X_i$  is a *leaf bag*.

Given a tree-decomposition of width  $\omega$  of  $G$ , a nice tree-decomposition of  $G$  with the same width can be computed in  $O(\omega \cdot n)$  time [10].

### 3 The $st$ -Orientation Problem Parameterized by Treewidth

In this section, we describe a fixed-parameter tractable algorithm for ST-ORIENTATION parameterized by treewidth. In fact, the algorithm we propose can solve a slightly more general problem. Namely, it does not assume that  $s$  and  $t$  are part of the input, but it looks for an  *$st$ -orientation* in which the source and the sink can be any pair of vertices of the input graph. However, if  $s$  and  $t$  are prescribed, a simple check can be added to the algorithm (we will highlight the crucial point in which the check is needed) to ensure this property.

Let  $G = (V, E)$  be an undirected graph. A *solution* of the ST-ORIENTATION problem is an orientation  $O$  of  $G$  such that  $D_O(G)$  is an  *$st$ -graph* with at most  $k$  transitive edges. Let  $(\mathcal{X}, T)$  be a tree-decomposition of  $G$  of width  $\omega$ . For a bag  $X_i \in \mathcal{X}$ , we denote by  $G[X_i]$  the



■ **Figure 3** (a) The directed edges  $wv$  and  $vu$  make all edges  $e_1, \dots, e_s$  transitive. (b) Each pair of directed edges  $w_p v$  and  $v u_q$ , for  $p \in [1, h]$  and  $q \in [1, s]$ , makes  $e$  transitive.

subgraph of  $G$  induced by the vertices of  $X_i$ , and by  $T_i$  the subtree of  $T$  rooted at  $X_i$ . Also, we denote by  $G_i$  the subgraph of  $G$  induced by all the vertices in the bags of  $T_i$ . We adopt a dynamic-programming approach performing a bottom-up traversal of  $T$ . The solution space is encoded into records associated with the bags of  $T$ , which we describe in the next section.

### 3.1 Encoding solutions

Before describing the records stored for each bag, we highlight the main challenges about how to encode the partial solutions computed throughout the course of the algorithm. Let  $v$  be a vertex introduced in a bag  $X_i$ . Adding  $v$  and its incident edges to a partial solution may either turn many (possibly linearly many) forgotten edges into transitive edges and/or it may make the same forgotten edge transitive with respect to arbitrarily many different paths. This is schematically illustrated in Figure 3, where  $X_i$  and its child bag  $X_j$  are highlighted by shaded regions. In Figure 3a,  $e_1, \dots, e_s$  are forgotten edges, i.e., edges in  $G_i$  but not in  $G[X_i]$ ; if we orient edge  $(u, v)$  from  $v$  to  $u$  and edge  $(v, w)$  from  $w$  to  $v$  all edges  $e_1, \dots, e_s$  become transitive. In Figure 3b,  $e$  is a forgotten edge, while  $u_1, \dots, u_s$  and  $w_1, \dots, w_h$  are vertices of bag  $X_j$ ; orienting the edges  $(w_p, v)$  from  $w_p$  to  $v$  ( $1 \leq p \leq h$ ) and the edges  $(v, u_q)$  from  $v$  to  $u_q$ , turns  $e$  into a transitive edge with respect to  $h \times s$  different paths. In case of Figure 3a the algorithm cannot afford reconsidering the forgotten edges as they can be arbitrarily many. In case of Figure 3b the algorithm should avoid counting  $e$  multiple times (for each newly created path). To overcome these issues, the algorithm guesses the edges that are transitive in a candidate solution and verifies that no other edge can become transitive during the bottom-up visit of  $T$ . This is done by suitable records, describe below.

Let  $O$  be a solution and consider a bag  $X_i \in \mathcal{X}$ . The *record*  $R_i$  of  $X_i$  that *encodes*  $O$  represents the interface of the solution  $O$  with respect to  $X_i$ . For ease of notation, the restriction of  $D_O(G)$  to  $G_i$  is denoted by  $D_i$ , and similarly the restriction to  $G[X_i]$  is  $D[X_i]$ . Record  $R_i$  stores the following information.

- $\mathcal{O}_i$  which is the orientation of  $D[X_i]$ .
- $\mathcal{A}_i$  which is the subset of the edges of  $D[X_i]$  that are transitive in  $D_O(G)$ . We call such edges *admissible transitive edges* or simply *admissible edges*. The edges of  $G_i$  not in  $\mathcal{A}_i$  are called *non-admissible*. We remark that an edge of  $\mathcal{A}_i$  may not be transitive in  $D_i$ .
- $\mathcal{P}_i$  which is the set of ordered pairs of vertices  $(a, b)$  such that: (i)  $a, b \in X_i$ , and (ii)  $D_i$  contains the path  $a \rightsquigarrow b$ .

- $\mathcal{F}_i$  which is the set of ordered pairs of vertices  $(a, b)$  such that: (i)  $a, b \in X_i$ , and (ii) connecting  $a$  to  $b$  with a directed path makes a non-admissible edge of  $D_i$  to become transitive.
- $c_i$  which is the *cost* of  $R_i$ , that is, the number of transitive edges in  $D_i$ . Note that  $c_i \geq |\mathcal{A}_i|$ .
- $\mathcal{S}_i$  which maps each vertex  $v \in X_i$  to a Boolean value  $\mathcal{S}_i(v)$  that is true if and only if  $v$  is a source in  $D_i$ . Analogously,  $\mathcal{T}_i$  maps each vertex  $v \in X_i$  to a Boolean value  $\mathcal{T}_i(v)$  that is true if and only if  $v$  is a sink in  $D_i$ .
- $\sigma_i$  which is a flag that indicates whether  $D_O(G)$  contains a source that belongs to  $G_i$  but not to  $X_i$ . Analogously,  $\tau_i$  is a flag that indicates whether  $D_O(G)$  contains a sink that belongs to  $G_i$  but not to  $X_i$ .

Observe that, for a bag  $X_i$ , different solutions  $O$  and  $O'$  of  $G$  may be encoded by the same record  $R_i$ . In this case,  $O$  and  $O'$  are *equivalent*. Clearly, this defines an equivalent relation on the set of solutions for  $G$ , and each record represents an equivalence class. The goal of the algorithm is to incrementally construct the set of records (i.e., the quotient set) for each bag rather than the whole set of solutions. More formally, for each bag  $X_i \in \mathcal{X}$ , we associate a set of records  $\mathcal{R}_i = \{R_i^1, \dots, R_i^h\}$ . While this is not essential for establishing fixed-parameter tractability, we further observe that if more records are equal except for their costs, it suffices to keep in  $\mathcal{R}_i$  the one whose cost is no larger than any other record. The next lemma easily follows.

► **Lemma 1** ( $\star$ ). *For a bag  $X_i$ , the cardinality of  $\mathcal{R}_i$  is  $2^{O(\omega^2)}$ . Also, each record of  $\mathcal{R}_i$  has size  $O(\omega^2)$ .*

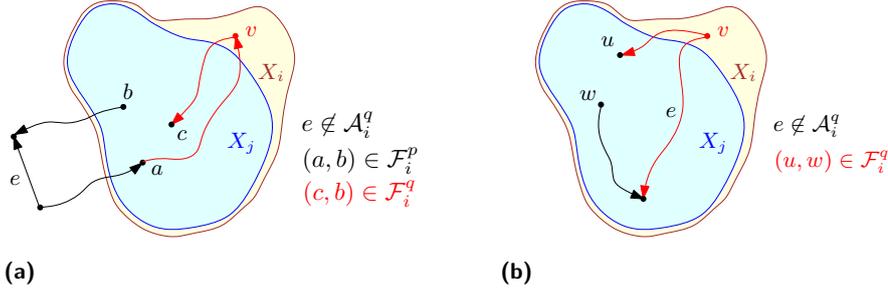
### 3.2 Description of the algorithm

We are now ready to describe our dynamic-programming algorithm over a nice tree-decomposition  $(\mathcal{X}, T)$  of the input graph  $G$ . Let  $X_i$  be the current bag visited by the algorithm. We compute the records of  $X_i$  based on the records computed for its child or children (if any). If the set of records of a bag is empty, the algorithm halts and returns a negative answer. We distinguish four cases based on the type of the bag  $X_i$ . Observe that, to index the records within  $\mathcal{R}_i$ , we added a superscript  $q \in [h]$  to each record, and we will do the same for all the record's elements.

**$X_i$  is a leaf bag.** We have that  $X_i$  is the empty set and  $\mathcal{R}_i$  consists of only one record, i.e.,  $\mathcal{R}_i = \{R_i^1 = \langle \emptyset, \emptyset, \emptyset, \emptyset, 0, \emptyset, \emptyset, \text{false}, \text{false} \rangle\}$ .

**$X_i$  is an introduce bag.** Let  $X_j = X_i \setminus \{v\}$  be the child of  $X_i$ . Initially,  $\mathcal{R}_i = \emptyset$ . Next, the algorithm exhaustively extends each record  $R_j^p \in \mathcal{R}_j$  to a set of records of  $\mathcal{R}_i$  as follows. Let  $\mathcal{O}_v$  be the set of all the possible orientations of the edges incident to  $v$  in  $G[X_i]$ , and similarly let  $\mathcal{A}_v$  be the set of all the possible subsets of the edges incident to  $v$  in  $G[X_i]$ . The algorithm considers all possible pairs  $(o, t)$  such that  $o \in \mathcal{O}_v$  and  $t \in \mathcal{A}_v$ . For each pair  $(o, t)$ , we proceed as follows.

1. Let  $q = |\mathcal{R}_i| + 1$ , the algorithm computes a candidate orientation  $\mathcal{O}_i^q$  of  $G[X_i]$  starting from  $\mathcal{O}_j^p$  and orienting the edges of  $v$  according to  $o$ .
2. Similarly, it computes the candidate set of admissible edges  $\mathcal{A}_i^q$  starting from  $\mathcal{A}_j^p$  and adding to it the edges in  $t$ .
3. Next, it sets the candidate cost  $c_i^q = c_j^p + |t|$ .



■ **Figure 4** Illustration of Step 5c of the algorithm when  $X_i$  is an introduce bag.

4. Let the *extension*  $\langle \mathcal{O}_i^q, \mathcal{A}_i^q, c_i^q \rangle$  be *valid* if:

- a.  $c_i^q \leq k$ ;
- b. there is no pair  $(a, b) \in \mathcal{P}_j^p$  so that  $bv, va \in D[X_i^q]$ ;
- c. there is no pair  $(a, b) \in \mathcal{F}_j^p$  so that  $av, vb \in D[X_i^q]$ .

Clearly, if an extension is not valid, the corresponding record cannot encode any solution; namely, condition (a) ensures that the candidate cost does not exceed  $k$ , condition (b) guarantees the absence of cycles, condition (c) guarantees that no non-admissible edge becomes transitive. Hence, if an extension is not valid, the algorithm discards it and continues with the next pair  $(o, t)$ .

5. Instead, if the extension is valid, the algorithm computes the record  $R_i^q = \langle \mathcal{O}_i^q, \mathcal{A}_i^q, \mathcal{P}_i^q, \mathcal{F}_i^q, c_i^q, \mathcal{S}_i^q, \mathcal{T}_i^q, \sigma_i^q, \tau_i^q \rangle$  of  $\mathcal{R}_i$ , where  $\sigma_i^q = \sigma_j^p, \tau_i^q = \tau_j^p$  (recall that  $X_j \subset X_i$ ). To complete the record  $R_i^q$ , it remains to compute  $\mathcal{S}_i^q, \mathcal{T}_i^q, \mathcal{P}_i^q$  and  $\mathcal{F}_i^q$ .

- a. For each vertex  $w \in X_j$ , we set  $\mathcal{S}_i^q(w) = \text{true}$  if and only if  $\mathcal{S}_j^p(w) = \text{true}$  and there is no edge of  $v$  oriented from  $v$  to  $w$  in  $D[X_i^q]$  (which would make  $w$  not a source anymore). Similarly, for each vertex  $w \in X_j$ , we set  $\mathcal{T}_i^q(w) = \text{true}$  if and only if  $\mathcal{T}_j^p(w) = \text{true}$  and there is no edge of  $v$  oriented from  $w$  to  $v$  in  $D[X_i^q]$ . Finally, we set  $\mathcal{S}_i^q(v) = \text{true}$  if and only if  $v$  is a source in  $D[X_i^q]$  (as by the definition), and we set  $\mathcal{T}_i^q(v) = \text{true}$  if and only if  $v$  is a sink in  $D[X_i^q]$ .
- b. We initially set  $\mathcal{P}_i^q = \emptyset$ . We recompute the paths from scratch as follows. We build an auxiliary digraph  $D^*$  which we initialize with  $D[X_i^q]$ . We then add to  $D^*$  the information about paths in  $\mathcal{P}_j^p$ . Namely, for each  $(a, b) \in \mathcal{P}_j^p$ , we add an edge  $ab$  to  $D^*$  (if it does not already exist). Once this is done, for each pair  $u, w \in X_i \times X_i$  for which there is a path  $u \rightsquigarrow w$  in  $D^*$ , we add the pair  $(u, w)$  to  $\mathcal{P}_i^q$ .
- c. Consider now  $\mathcal{F}_i^q$ . We initially set  $\mathcal{F}_i^q = \mathcal{F}_j^p$ . Observe that the addition of  $v$  might have created new pairs of vertices that should belong to  $\mathcal{F}_i^q$ . Namely, for each pair  $(a, b) \in \mathcal{F}_j^p$ , we verify what are the vertices  $c$  such that  $D[X_i^q]$  contains a path  $a \rightsquigarrow c$  while  $D[X_j^p]$  does not (observe that  $a \rightsquigarrow c$  contains  $v$ , possibly  $c = v$ ); for each such vertex, we add  $(c, b)$  to  $\mathcal{F}_i^q$ . See Figure 4a for an illustration. Similarly, we verify what are the vertices  $d$  such that  $D[X_i^q]$  contains a path  $d \rightsquigarrow b$  while  $D[X_j^p]$  does not (again  $d \rightsquigarrow b$  contains  $v$ , possibly  $d = v$ ); for each such vertex, we add  $(a, d)$  to  $\mathcal{F}_i^q$ . Finally, we consider all the edges incident to  $v$  and that are not in  $\mathcal{A}_i^q$ . These edges are not admissible and we should further update  $\mathcal{F}_i^q$  accordingly. This can be done as follows: we consider each edge incident to  $v$  not in  $\mathcal{A}_i^q$ , for each such an edge  $e$  we verify what are the pairs of vertices in  $X_i$  (including  $e$ 's endpoints) such that connecting them with a path makes  $e$  transitive, we add such pairs to  $\mathcal{F}_i^q$  if not already present. See Figure 4b for an illustration.

**$X_i$  is a forget bag.** Let  $X_j = X_i \cup \{v\}$  be the child of  $X_i$ . The algorithm computes  $\mathcal{R}_i$  by exhaustively merging records of  $\mathcal{R}_j$  as follow.

1. For each  $R_j^p \in \mathcal{R}_j$ , we remove from  $\mathcal{O}_j^p$  and  $\mathcal{A}_j^p$  all the edges incident to  $v$  and from  $\mathcal{P}_j^p$  and  $\mathcal{F}_j^p$  all the pairs where one of the vertices is  $v$ . Observe that due to this operation, there might now be records that are identical except possibly for their costs. Among them, we only keep one record whose cost is no larger than any other record.
2. Let  $R_j^p$  be a record of  $\mathcal{R}_j$  that was not discarded by the procedure above. If  $\mathcal{S}_j^p(v) \wedge \sigma_j^p$ , we discard  $R_j^p$  (because the encoded orientation would contain two sources), else we set  $\sigma_j^p = \text{true}$  (because  $v$  is a source). Similarly, if  $\mathcal{T}_j^p(v) \wedge \tau_j^p$ , we discard  $R_j^p$ , else we set  $\tau_j^p = \text{true}$ . At this point, if the record has not been discarded yet and vertices  $s$  and  $t$  are prescribed, we can add the following check. If  $\mathcal{S}_j^p(v) \wedge \sigma_j^p$ , then  $v$  is a source, hence if  $v \neq s$ , we discard the record. Analogously, if  $\mathcal{T}_j^p(v) \wedge \tau_j^p$ , then  $v$  is a sink, hence if  $v \neq t$ , we discard the record.
3. Finally, we remove from  $\mathcal{S}_j^p$  and  $\mathcal{T}_j^p$  the values  $\mathcal{S}_j^p(v)$  and  $\mathcal{T}_j^p(v)$ .
4. All the records that have not been discarded and have been updated according to the above procedure are added to  $\mathcal{R}_i$ .

**$X_i$  is a join bag.** Let  $X_j = X_{j'}$  be the two children of  $X_i$ . The algorithm computes  $\mathcal{R}_i$  by exhaustively checking if a pair of records, one from  $X_j$  and one from  $X_{j'}$ , can be merged together. For each pair  $R_j^p$  and  $R_{j'}^{p'}$ , we proceed as follows.

1. We initially set  $\mathcal{R}_i = \emptyset$ . The two records  $R_j^p$  and  $R_{j'}^{p'}$  are *mergeable* if:
  - a.  $\mathcal{O}_j^p = \mathcal{O}_{j'}^{p'}$ ;
  - b.  $\mathcal{A}_j^p = \mathcal{A}_{j'}^{p'}$ ;
  - c.  $c_j^p + c_{j'}^{p'} - |\mathcal{A}_j^p| \leq k$ ;
  - d. there is no pair  $(a, b) \in \mathcal{P}_j^p$  such that  $(b, a) \in \mathcal{P}_{j'}^{p'}$ ;
  - e. there is no pair  $(a, b) \in \mathcal{P}_j^p$  such that  $(a, b) \in \mathcal{F}_{j'}^{p'}$ ;
  - f. there is no pair  $(a, b) \in \mathcal{P}_{j'}^{p'}$  such that  $(a, b) \in \mathcal{F}_j^p$ ;
  - g.  $\neg(\sigma_j^p \wedge \sigma_{j'}^{p'})$ ;
  - h.  $\neg(\tau_j^p \wedge \tau_{j'}^{p'})$ .

Conditions **a-b** are obviously necessary to merge the records. Condition **c** guarantees that the number of transitive edges (avoiding double counting the admissible edges in  $X_i$ ) is at most  $k$ . Condition **d** guarantees the absence of cycles. Conditions **e-f** guarantee that no non-admissible edge becomes transitive. Conditions **g-h** guarantee that the resulting orientation contains at most one source and one sink. If the two records are not mergeable, we discard the pair and proceed with the next one. Otherwise we create a new record  $R_i^q$ , with  $q = |\mathcal{R}_i| + 1$ , and continue to the next step.

2. Based on the previous discussion, we can now compute  $R_i^q$  as follows:
  - a.  $\mathcal{O}_i^q = \mathcal{O}_j^p$ ;
  - b.  $\mathcal{A}_i^q = \mathcal{A}_j^p$ ;
  - c.  $c_i^q = c_j^p + c_{j'}^{p'} - |\mathcal{A}_j^p|$ ;
  - d. For each pair  $(a, b)$  of vertices of  $X_i$ , we add it to  $\mathcal{P}_i^q$  if it is contained in  $\mathcal{P}_j^p$  or in  $\mathcal{P}_{j'}^{p'}$ .
  - e. For each pair  $(a, b)$  of vertices of  $X_i$ , we add it to  $\mathcal{F}_i^q$  if  $\mathcal{F}_j^p(a, b) \vee \mathcal{F}_{j'}^{p'}(a, b)$ .
  - f. For each vertex  $v$  of  $X_i$ , we set  $\mathcal{S}_i^q(v) = \mathcal{S}_j^p(v) \wedge \mathcal{S}_{j'}^{p'}(v)$ ;
  - g. For each vertex  $v$  of  $X_i$ , we set  $\mathcal{T}_i^q(v) = \mathcal{T}_j^p(v) \wedge \mathcal{T}_{j'}^{p'}(v)$ ;
  - h.  $\sigma_i^q = \sigma_j^p \vee \sigma_{j'}^{p'}$ ;
  - i.  $\tau_i^q = \tau_j^p \vee \tau_{j'}^{p'}$ .

The next lemma establishes the correctness of the algorithm.

► **Lemma 2.** *Graph  $G$  admits a solution for ST-ORIENTATION if and only if the algorithm terminates after visiting the root of  $T$ . Also, the algorithm outputs a solution, if any.*

**Proof.** ( $\rightarrow$ ) Suppose that the algorithm terminates after visiting the root bag  $X_\rho$  of  $T$ . We reconstruct a solution  $O$  of  $G$  as follows. We can assume that our algorithm stores additional pointers for each record (a common practice in dynamic programming), such that each record has a single outgoing pointer (and potentially many incoming pointers). Consider a record  $R_i^q$  of a bag  $X_i$ . If  $X_i$  is an introduce bag, there is only one record  $R_j^p$  of the child bag  $X_j$  from which  $R_i^q$  was generated and the pointer links  $R_i^q$  and  $R_j^p$ . If  $X_i$  is forget bag, there might be multiple records that have been merged into  $R_i^q$  and in this case the pointer link  $R_i^q$  with one of these records with minimum cost. If  $X_i$  is a join bag, there are two mergeable records  $R_j^p$  and  $R_{j'}^{p'}$  that have been merged together, and the pointer links  $R_i^q$  to  $R_j^p$  and  $R_{j'}^{p'}$ . With these pointers at hand, we can apply a top-down traversal of  $T$ , starting from the single (empty) record of the root bag  $X_\rho$  and reconstruct the corresponding orientation  $O$ . Namely, when visiting an introduce bag and the corresponding record, we orient the edges of the introduced vertex  $v$  according to the orientation  $O_v$  defined by the record.

We now claim that  $D_O(G)$  is an *st*-graph with at most  $k$  transitive edges. Suppose first, for a contradiction, that  $D_O(G)$  contains more than one source. Let  $s$  and  $s'$  be two sources of  $D_O(G)$ . Then  $S_i^q(s) = \text{false}$  in the bag  $X_i$  in which  $s$  has been forgotten, and similarly for  $S_i^q(s')$ . This is however not possible by construction of  $S_i^q$ . Thus, either the record  $R_i^q$  has been discarded because  $S_j^p(v) \wedge \sigma_j^p$  (see item **2** when  $X_i$  is a forget bag) or  $\sigma_j^p = \text{false}$ . The first case contradicts the fact that  $R_i^q$  is a record used to reconstruct  $O$ . The second case implies that  $s'$  has not been encountered; however, in this latter case the algorithm sets  $\sigma_j^p = \text{true}$ , hence some descendant record will be discarded as soon as  $s'$  is forgotten, again contradicting the fact that we are considering records with pointers up to the root bag. A symmetric argument shows that  $D_O(G)$  contains a single sink. We next argue that  $D_O(G)$  is acyclic. Suppose, again for a contradiction, that  $D_O(G)$  contains a cycle. In particular, the cycle was created either in an introduce bag or in a join bag. In the former case, let  $v$  be the last vertex of this cycle that has been introduced in a bag  $X_i$ . Let  $a, b$  be the neighbors of  $v$  that are part of the cycle, and w.l.o.g. assume that the edges are  $va$  and  $bv$ . It must be  $\mathcal{P}_i^q$  does not contain the pair  $(a, b)$ , otherwise we would have discarded this particular orientation for the edges incident to  $v$  (see item **4.b** when  $X_i$  is an introduce bag). On the other hand, one easily verifies that when introducing a vertex  $v$ , all the new paths involving  $v$  are computed from scratch (see item **5.b** when  $X_i$  is an introduce bag), and, similarly, when joining two bags, the existence of a path in one of the two bags is correctly reported in the new record (see item **2.d** when  $X_i$  is a join bag). If the cycle was created in a join bag the argument is analogous, in particular, observe that we verify that there is no path contained in the record of one of the child bags such that the same path with reversed direction exists in the record of the other child bag (see item **1.d** when  $X_i$  is a join bag). We conclude this direction of the proof by showing that  $D_O(G)$  contains at most  $k$  transitive edges. Observe first that the cost of the record ensures that at most  $k$  edges of  $G$  are part of some set of admissible edges. Suppose, for a contradiction, that  $D_O(G)$  contains more than  $k$  transitive edges. Then there is a bag  $X_i$  and a record  $R_i^q$  in which a non-admissible edge became transitive. Also,  $X_i$  is either an introduce or a join bag. If  $X_i$  introduced a vertex  $v$ , observe that all the newly introduced edges are incident to  $v$ . On the other hand, the algorithm discarded the orientations of the edges of  $v$  for which there is a pair  $(a, b) \in \mathcal{F}_j^p$  (with  $X_j$  being the child of  $X_i$ ) so that  $av, vb \in D[X_i^q]$  (see item **4.c** when

## 18:10 On the Parameterized Complexity of Computing $st$ -Orientations

$X_i$  is an introduce bag). Then either the orientation was discarded, which contradicts the fact that we are considering a record used to build the solution, or  $\mathcal{F}_j^p$  missed the pair  $(a, b)$ . Again one verifies this second case is not possible, because the new pairs that are formed in an introduce bag are correctly identified (see item 5.c when  $X_i$  is an introduce bag) by the algorithm and similarly for join bags (see item 2.e when  $X_i$  is a join bag). If  $X_i$  is a join bag, the argument is analogous, in particular, we verified that there is no path in one of the two child records that makes transitive a non-admissible edge in the other child record (see items 1.e and 1.f when  $X_i$  is a join bag). This concludes the first part of the proof.

( $\leftarrow$ ) It remains to prove that, if  $G$  admits a solution  $O$ , then the algorithm terminates after visiting the root  $X_\rho$  of  $T$ . If this were not the case, there would be a bag  $X_i$  of  $T$  and a candidate record that encodes  $O$ , such that the record has been incorrectly discarded by the algorithm; we show that this is not possible. Suppose first that  $X_i$  is an introduce bag. Then a candidate record is discarded if the cost exceeds  $k$ , or if a cycle is created, or if a non-admissible edge becomes transitive (see the conditions of item 4 when  $X_i$  is an introduce bag). In all cases the candidate record does not encode a solution. If  $X_i$  is a forget bag, we may discard a candidate record if it is identical to another but has a non-smaller cost (see item 1 when  $X_i$  is a forget bag). Hence we always keep a record that either encodes the solution at hand or a solution with fewer transitive edges but with exactly the same interface at  $X_i$ . Also, we may discard a record if the forgotten vertex  $v$  is a source and  $G_i$  already contains a source (see item 2 when  $X_i$  is a forget bag). This is correct, because no further edge can be added to  $v$  after it is forgotten. A symmetric argument holds for the case in which a record is discarded due to  $v$  being a sink. Finally, if  $X_i$  is a join bag, pairs of records of its children bags are discarded if not mergeable (see the conditions of item 1 when  $X_i$  is a join bag). One easily verifies that failing one of the conditions for mergeability implies that the record does not encode a solution (see also the discussion after item 1).  $\blacktriangleleft$

The next theorem summarizes our contribution.

**► Theorem 3 ( $\star$ ).** *Given an input graph  $G = (V, E)$  of treewidth  $\omega$  and an integer  $k \geq 0$ , there is an algorithm that either finds a solution of ST-ORIENTATION or reject the input in time  $2^{O(\omega^2)} \cdot n$ .*

### 4 The Complexity of the Non-Transitive $st$ -Orientation Problem for Graphs of Bounded Diameter and Bounded Degree

We begin by recalling the special case of ST-ORIENTATION considered in [1]. An  $st$ -orientation  $O$  of a graph  $G$  is *non-transitive* if  $D_O(G)$  does not contain transitive edges.

NON-TRANSITIVE ST-ORIENTATION (NT-ST-ORIENTATION)

**Input:** An undirected graph  $G = (V, E)$ , and two vertices  $s, t \in V$ .

**Output:** An non-transitive  $st$ -orientation  $O$  of  $G$  such that vertices  $s$  and  $t$  are the source and sink of  $D_O(G)$ , respectively.

The hardness proof of NT-ST-ORIENTATION in [1] exploits a reduction from NOT-ALL-EQUAL 3-SAT (NAE-3-SAT) [13]. Recall that the input of NAE-3-SAT is a pair  $\langle X, \varphi \rangle$  where  $X$  is a set of boolean variables and  $\varphi$  is a set of clauses, each composed of three literals out of  $X$ , and the problem asks for an assignment of the variables in  $X$  so that each clause in  $\varphi$  is composed of at least one true variable and one false variable.

In this section, we show that NT-ST-ORIENTATION is NP-hard even for graphs of bounded diameter and for graphs of bounded vertex degree that are subdivisions of triconnected graphs. To prove our results, we first summarize the construction used in [1].

#### 4.1 A Glimpse into the Hardness Proof of NT-st-Orientation

The construction in [1] adopts three types of gadgets, which we recall below. Given an edge  $e$  of a digraph  $D$  such that  $e$  has an end-vertex  $v$  of degree 1, we say that  $e$  *enters*  $D$  if it is outgoing with respect to  $v$ , and we say that  $e$  *exits*  $D$  otherwise. Similarly, given a directed edge  $e = uv$ , we say that  $e$  *exits*  $u$  and that  $e$  *enters*  $v$ .

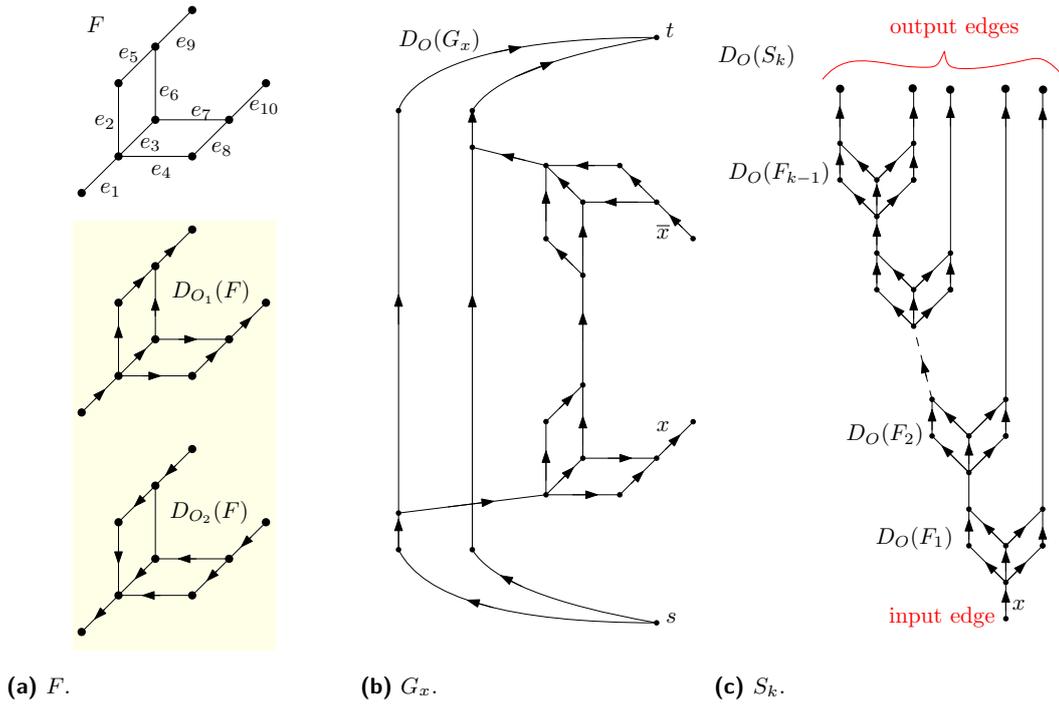
- The *fork gadget*  $F$  is depicted in Figure 5a. See Lemma 1 of [1]. Namely, if  $F$  does not contain  $s$  or  $t$  (the source and sink prescribed in the input), then in any non-transitive orientation  $O$  of a graph  $G$  containing  $F$ , either  $e_1$  enters  $F$  and  $e_9, e_{10}$  exit  $F$ , or vice versa. Figure 5a depicts  $F$ ,  $D_{O_1}(F)$  and  $D_{O_2}(F)$ , where  $O_1$  and  $O_2$  are the two  $st$ -orientations admitted by  $F$ .
- The *variable gadget*  $G_x$  associated to a variable  $x \in X$  is shown in Figure 5b; observe it contains the designated vertices  $s$  and  $t$ . Its crucial property is stated in Lemma 2 of [1]. Namely, in any non-transitive  $st$ -orientation  $O$  of a graph  $G$  containing  $G_x$ , either  $x$  exists  $G_x$  and  $\bar{x}$  enters  $G_x$ , or vice-versa.
- The *split gadget*  $S_k$  is shown in Figure 5c; it consists of  $k-1$  fork gadgets chained together, for some fixed  $k > 0$ . The crucial property of this gadget is described in Lemma 3 of [1]. Namely, in any non-transitive  $st$ -orientation  $O$  of a graph  $G$  containing  $S_k$ , either  $x$  (the *input edge* of  $S_k$ ) enters  $S_k$  and the edges  $e_9$  and  $e_{10}$  of the fork gadgets  $F_1, \dots, F_{k-1}$  incident to one degree-1 vertex (the *outgoing edges* of  $S_k$ ) exit  $S_k$ , or vice-versa.

Given an instance  $\langle X, \varphi \rangle$  of NAE-3-SAT, the instance  $\langle G_\varphi, s, t \rangle$  of NT-ST-ORIENTATION is constructed as follow. For each  $x \in X$  we add  $G_x$  and two split gadgets  $S_k$  and  $S_{\bar{k}}$ , where  $k$  (resp.  $\bar{k}$ ) is the number of clauses where  $x$  appears in its non-negated (resp. negated) form (edges  $x$  and  $\bar{x}$  are the input edges of  $S_k$  and  $S_{\bar{k}}$ , respectively). Finally, for each clause  $c = (x_1, x_2, x_3) \in \varphi$ , we add a vertex  $c$  that is incident to an output edge of the split gadget of each of its variables. See Figure 6b, where the non-dashed edges and all the vertices with the exception of  $g$  define  $G_\varphi$ . It can be shown that  $\langle X, \varphi \rangle$  is a yes-instance of NAE-3-SAT if and only if  $\langle G_\varphi, s, t \rangle$  is a yes-instance of NT-ST-ORIENTATION [1].

#### 4.2 Hardness for Graphs of Bounded Diameter

Given an undirected graph  $G$ , the *distance* between two vertices of  $G$  is the length of any shortest path connecting them. The *diameter* of  $G$  is the maximum distance over all pairs of vertices of the graph. We now adapt the construction in Section 4.1 to show that NT-ST-ORIENTATION remains NP-hard also for graphs of bounded diameter. We define the *extended fork gadget* by adding an edge  $e_{11}$  to the fork gadget (see Figure 6a).

**Construction of  $H_\varphi$ .** Given an instance  $\langle X, \varphi \rangle$  of NAE-3-SAT and the instance  $\langle G_\varphi, s, t \rangle$  of NT-ST-ORIENTATION computed as described in Section 4.1, we define  $\langle H_\varphi, s, t \rangle$  as follows. We first set  $H_\varphi = G_\varphi$ . Then, we add a vertex  $g$  to  $H_\varphi$  and an edge  $(g, f)$  for each vertex  $f$  belonging to a fork  $F$  of  $H_\varphi$  and incident to the corresponding edges  $e_3, e_6$ , and  $e_7$ . Also, we add edges  $(g, t)$  and  $(s, g)$ , and we subdivide each of them once. See Figure 6b (the non-dashed edges and all the vertices with the exception of  $g$  define  $G_\varphi$ ).



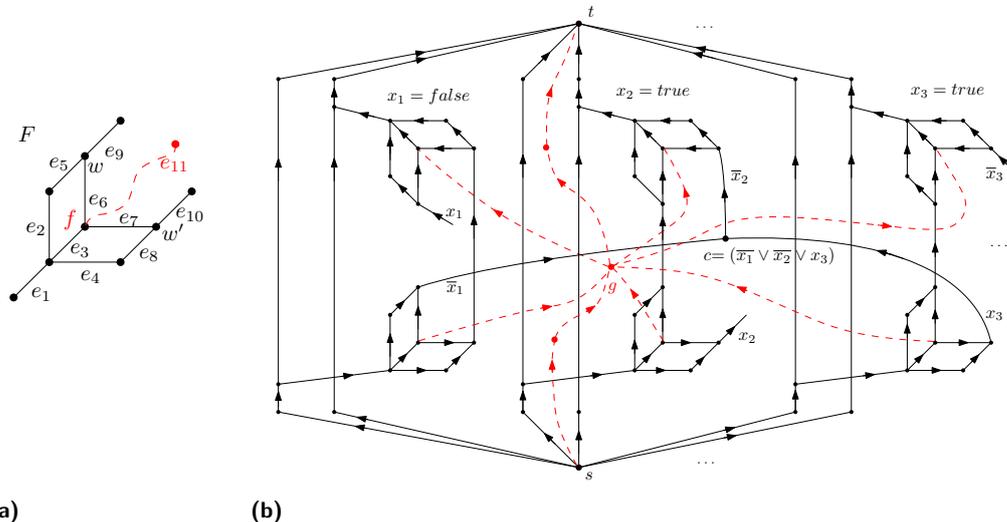
■ **Figure 5** (a) The fork gadget  $F$  and its two possible non-transitive  $st$ -orientations. (b) The variable gadget  $D_O(G_x)$  associated to  $x \in X$ , where  $O$  is one of its two possible orientations. (c) The split gadget  $D_O(S_k)$  associated to  $x$ , where  $O$  is one of its two possible orientations.

► **Theorem 4** ( $\star$ ). *NT-ST-ORIENTATION is NP-hard for graphs of diameter at most 6.*

**Proof sketch.** We construct  $H_\varphi$  as described above. Observe that any vertex of  $G$  is at distance at most 3 to  $g$ , hence  $H_\varphi$  has diameter at most 6. We show that a non-transitive  $st$ -orientation of  $G_\varphi$  corresponds to a non-transitive  $st$ -orientation of  $H_\varphi$  ( $\rightarrow$ ) and vice versa ( $\leftarrow$ ).

$(\rightarrow)$  Given a non-transitive  $st$ -orientation  $O'$  of  $G_\varphi$ , we construct an  $st$ -orientation  $O$  of  $H_\varphi$  by extending  $O'$  as follow. We orient the four edges of  $H_\varphi \setminus G_\varphi$  connecting  $s$  to  $t$  such that the path is directed from  $s$  to  $g$ . For each other edge  $e$ , which is incident to  $g$ , we orient it so that  $e$  enters  $g$  if and only if  $e$  is the edge incident to an extended fork gadget whose corresponding edge  $e_1$  is an entering edge. See Figure 6b. For each two vertices  $a, b \in D_O(H_\varphi)$ , there is no path  $a \rightsquigarrow b$  so that  $ag, gb \in D_O(H_\varphi)$ . Hence, since  $D_{O'}(G_\varphi)$  has no cycle, also  $D_O(H_\varphi)$  has no cycle. Consequently,  $O$  is an acyclic orientation with  $s$  and  $t$  being its single source and sink, respectively. We now show that it does not contain transitive edges. Let  $e = ab$  be any edge of  $D_O(H_\varphi)$ . We have that any path from  $s \rightsquigarrow t$  containing  $g$  either contains edges incident to degree-2 vertices or edges  $e_1, e_3$ , and  $e_{11}$  of an extended fork gadget. All these edges have endpoints which are not adjacent by construction. Hence, there is no path  $a \rightsquigarrow b$  containing  $g$  and, since  $O'$  is non-transitive,  $e$  is not transitive in  $D_O(H_\varphi)$ .

$(\leftarrow)$  This direction is based on the observation that, given an extended fork gadget  $F$ , in any non-transitive  $st$ -orientation  $O$  of  $G$ , either  $e_3$  enters  $f$  and  $e_6, e_7$  exit  $f$  or vice versa.  $\blacktriangleleft$



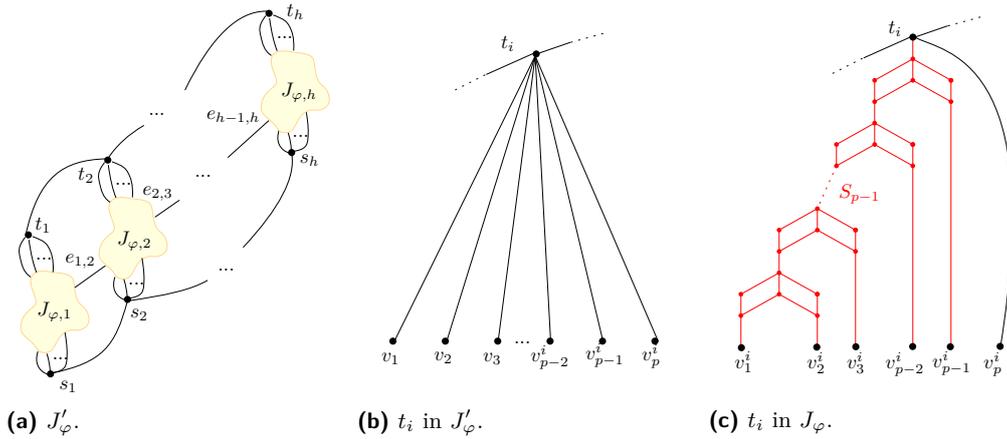
■ **Figure 6** (a) A fork gadget  $F$  extended with edge  $e_{11}$ . (b) Graphs  $D_{O'}(G_\varphi)$ , defined by the non-dashed edges, and graph  $D_O(H_\varphi)$ , obtained from  $G$  by adding  $g$  and the dashed edges.  $O'$  and  $O$  are non-transitive  $st$ -orientations of  $G_\varphi$  and  $H_\varphi$ , respectively.  $O$  is obtained by extending  $O'$ .

### 4.3 Hardness Subdivisions of Triconnected Graphs with Bounded Degree

We prove now that NT-ST-ORIENTATION is NP-hard even if  $G$  is a 4-graph, i.e., the degree of each vertex is at most 4, and, in addition, it is a subdivision of a triconnected graph.

**Construction of  $J_\varphi$ .** Given an instance  $\langle X, \varphi \rangle$  of NAE-3-SAT and the instance  $\langle G_\varphi, s, t \rangle$  of NT-ST-ORIENTATION computed as described in Section 4.1, we compute  $\langle J_\varphi, s, t \rangle$  as follows. We remove  $s$  and  $t$  from  $J'_\varphi = G_\varphi$ . We obtain a disconnected graph whose connected components are  $J_{\varphi,1}, \dots, J_{\varphi,h}$ . We add a vertex  $s_i$  and a vertex  $t_i$  to each  $J_{\varphi,i}$  (which will play the role of local sources and sinks for each component). Next, for each  $i \in [1, h-1]$ : (i) We add the edge  $(s_i, s_{i+1})$  and  $(t_{i+1}, t_i)$ ; (ii) We add an edge  $e_{i+1,i}$  incident to a vertex identified as the  $f$ -vertex of a fork gadget of  $J_{\varphi,i+1}$  and to a vertex identified as the  $f$ -vertex of a fork gadget of  $J_{\varphi,i}$ . We denote by  $J'_\varphi$  the obtained graph; see Figure 7a for a schematic illustration. For each  $J_{\varphi,i}$  ( $i \in [1, h]$ ) of  $J'_\varphi$ , the only vertices having degree higher than 4 are  $s_i$  and  $t_i$ . For each  $i \in [1, h]$ , we proceed as follow. We first consider  $t_i$ . If  $t_i$  has degree  $p \leq 4$ , we do nothing. Otherwise, if  $p \geq 5$ , we proceed as follows: (i) We consider  $p-1$  edges incident to  $t_i$  and to vertices of  $J_{\varphi,i}$  and we remove them; (ii) We connect the endpoints  $v_1^i, \dots, v_p^i$  of the removed edges that are not  $t_i$  to a split gadget  $S_{p-1}$  and  $t_i$  to the input edge of  $S_{p-1}$ . Figure 7b depicts  $t_i$  ( $i \in [1, h]$ ) and its neighborhood  $\{v_1^i, \dots, v_p^i\}$  in  $J'_\varphi$  and Figure 7c depicts how  $t_i$  is connected to the vertices  $v_1^i, \dots, v_p^i$  after the above operation. We perform a symmetric operation on  $s_i$ . The resulting graph is denoted by  $J_\varphi$ , and it has vertex degree at most four by construction.

► **Theorem 5** ( $\star$ ). *NT-ST-ORIENTATION is NP-hard for 4-graphs that are subdivisions of triconnected graphs.*



**Figure 7** (a) Schematic representation of graph  $J'_\varphi$ . (b-c) How vertex  $t_i$  is connected to its neighbourhood in (b)  $J'_\varphi$  and (c)  $J_\varphi$ .

## 5 Open Problems

Several interesting open problems stem from our research. Among them:

- Is there an FPT-algorithm for the ST-ORIENTATION problem parameterized by treewidth running in  $2^{o(\omega^2)} \cdot \text{poly}(n)$  time?
- Does ST-ORIENTATION parameterized by treedepth admit a polynomial kernel?
- We have shown that finding non-transitive  $st$ -orientations is NP-hard for graphs of vertex degree at most four. On the other hand, the problem is trivial for graphs of vertex degree at most two. What is the complexity of the problem for vertex degree at most three? Similarly, one can observe that the problem is easy for graphs of diameter at most two, while it remains open the complexity for diameter in the range  $[3, 5]$ .

---

## References

- 1 Carla Binucci, Walter Didimo, and Maurizio Patrignani.  $st$ -orientations with few transitive edges. In Patrizio Angelini and Reinhard von Hanxleden, editors, *GD 2022*, volume 13764 of *LNCS*, pages 201–216. Springer, 2022. doi:10.1007/978-3-031-22203-0\_15.
- 2 Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, Giacomo Ortali, and Tommaso Piselli. On the parameterized complexity of computing  $st$ -orientations with few transitive edges. *CoRR*, abs/2306.03196, 2023. arXiv:2306.03196.
- 3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.
- 5 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
- 6 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.
- 7 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
- 8 Shimon Even and Robert Endre Tarjan. Computing an  $st$ -numbering. *Theoretical Computer Science*, 2(3):339–344, 1976. doi:10.1016/0304-3975(76)90086-4.

- 9 Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs, Methods and Models*, volume 2025 of *LNCS*. Springer, 2001. doi:10.1007/3-540-44969-8.
- 10 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.
- 11 A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium.*, pages 215–232. Gordon and Breach, 1967.
- 12 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- 13 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. Association for Computing Machinery, 1978. doi:10.1145/800133.804350.



# Distributed CONGEST Algorithm for Finding Hamiltonian Paths in Dirac Graphs and Generalizations

Noy Biton 

Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Reut Levi  

Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Moti Medina  

Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

---

## Abstract

We study the problem of finding a Hamiltonian cycle under the promise that the input graph has a minimum degree of at least  $n/2$ , where  $n$  denotes the number of vertices in the graph. The classical theorem of Dirac states that such graphs (a.k.a. Dirac graphs) are Hamiltonian, i.e., contain a Hamiltonian cycle. Moreover, finding a Hamiltonian cycle in Dirac graphs can be done in polynomial time in the classical centralized model.

This paper presents a randomized distributed CONGEST algorithm that finds w.h.p. a Hamiltonian cycle (as well as maximum matching) within  $O(\log n)$  rounds under the promise that the input graph is a Dirac graph. This upper bound is in contrast to general graphs in which both the decision and search variants of Hamiltonicity require  $\tilde{\Omega}(n^2)$  rounds, as shown by Bachrach et al. [PODC'19].

In addition, we consider two generalizations of Dirac graphs: Ore graphs and Rahman-Kaykobad graphs [IPL'05]. In Ore graphs, the sum of the degrees of every pair of non-adjacent vertices is at least  $n$ , and in Rahman-Kaykobad graphs, the sum of the degrees of every pair of non-adjacent vertices plus their distance is at least  $n + 1$ . We show how our algorithm for Dirac graphs can be adapted to work for these more general families of graphs.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis

**Keywords and phrases** the CONGEST model, Hamiltonian Path, Hamiltonian Cycle, Dirac graphs, Ore graphs, graph-algorithms

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.19

**Related Version** *Full Version*: <https://arxiv.org/abs/2302.00742>

**Funding** *Noy Biton*: The author was supported by the Israel Science Foundation under Grant 1867/20.

*Reut Levi*: The author was supported by the Israel Science Foundation under Grant 1867/20.

*Moti Medina*: The author was supported by the Israel Science Foundation under Grant 867/19.

## 1 Introduction

The Hamiltonian path and Hamiltonian cycle problems are fundamental in computer science and appeared in Karp's 21 NP-complete problems [15]. A Hamiltonian path is a path that visits every vertex in the graph exactly once and a Hamiltonian cycle is a cycle that visits every vertex in the graph exactly once. We say that a graph is *Hamiltonian* if it contains a Hamiltonian cycle.

While the problem is hard in general (assuming  $P \neq NP$ ), for some classes of graphs, it is guaranteed that all their members are Hamiltonian. In particular, the classical theorem of Dirac states that every graph in which the minimum degree is at least  $n/2$  is Hamiltonian



© Noy Biton, Reut Levi, and Moti Medina;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



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

(we refer to graphs that satisfy this condition as DIRAC graphs). This condition is tight in the sense that if we are only guaranteed that the minimum degree is at least  $\alpha n$  for any  $0 < \alpha < 1/2$ , then the problem remains NP-complete [10].

In FOCS'87 Goldberg proposed the question of whether there is an NC-algorithm for finding a Hamiltonian cycle in DIRAC graphs.<sup>1</sup> This question was answered affirmatively by Dahlhaus et al. [9, 10], who gave a fast parallel algorithm on CREW-PRAM to find a Hamiltonian cycle in Dirac graphs. Their algorithm works in  $O(\log^4 n)$  parallel time and uses  $O(n + m)$  number of processors where  $m$  denotes the number of edges of the graphs.

Aside from the theoretical appeal of the problem, finding a Hamiltonian cycle in a graph also provides us with a maximum matching of the graph (and even perfect matching when  $n$  is even). Therefore, it is natural to ask whether the algorithm of Dahlhaus et al. [10] can be translated to the CONGEST model. To this end, one may attempt to use the PRAM simulation of Lotker, Patt-Shamir, and Peleg [19] for diameter-2 graphs (which applies for DIRAC graphs). However, this simulation is only valid when the number of processors is linear in the number of vertices in the graph. Another attempt is to use the more general transformation of Ghaffari and Li [13] that provide a randomized CONGEST algorithm that simulates any CRCW-PRAM algorithm that uses  $2m$  processors, runs in  $T$  parallel rounds, and operates on the input graph  $G$  that is stored in the PRAM's shared memory. The round complexity of the attained CONGEST algorithm is  $T \cdot \tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n})}$ , where  $\tau_{\text{mix}}(G)$  is the mixing-time of  $G$ . Thus even for constant mixing time, this yields a simulation of the algorithm of [10] in CONGEST with  $2^{O(\sqrt{\log n})}$  rounds. Moreover, since the mixing time of Dirac's graph can be  $\Theta(n)^2$ , the round complexity of this simulation can be super-linear in  $n$ . Consequently, our goal is to improve upon this round complexity by directly designing an algorithm for the CONGEST model. Indeed we provide an algorithm with an exponential improvement in the round complexity. Specifically, our algorithm for finding a Hamiltonian cycle in Dirac graphs runs in  $O(\log n)$  rounds. When the algorithm terminates, each vertex outputs the identifier of the vertex that is before it and the vertex that is after it on the cycle<sup>3</sup>.

In the CONGEST model, it is standard to assume that the processors have unbounded computational power. Therefore, one may wonder whether finding a Hamiltonian cycle in general graphs in  $o(n^2)$  rounds is possible. It was recently shown by Bachrach et al. [1] that even the problem of testing Hamiltonicity in the CONGEST model [21] requires  $\tilde{\Omega}(n^2)$  rounds. Therefore, it is natural to focus on restricted families of graphs such as DIRAC graphs and their generalizations. Since the classical result of Dirac, there have been many generalizations of Dirac's theorem (see [17] and references therein), e.g., graph families that are defined by sufficient conditions on degrees, neighborhoods, and other graph parameters [20, 2, 5, 18, 6, 7, 12, 11]. The first important generalization of Dirac's theorem is by Ore [20] who proved that every graph in which the sum of degrees of each pair of non-adjacent vertices is at least  $n$  is Hamiltonian. A more recent generalization, which also generalizes Ore's theorem, and allows graphs with less edges, is by Rahman and Kaykobad [22] who proved that every graph in which the sum of degrees of each pair of non-adjacent vertices plus their distance is at least  $n + 1$  has a Hamiltonian path. We refer to graphs that satisfy these conditions as ORE

<sup>1</sup> See [10] and [23] for more details.

<sup>2</sup> Consider a Dirac graph, over  $n$  vertices, which is composed of two cliques of size  $n/2$  with a perfect matching between the cliques.

<sup>3</sup> We note that although the input graph is undirected, the outputs of the vertices yield an oriented Hamiltonian cycle (or path).

graphs and RK graphs, respectively. We prove that our distributed CONGEST algorithm and its analysis can be adapted (without changing the round complexity asymptotically) for these generalizations of DIRAC graphs as well.

## Our Results

Our main result is stated in the following theorem.

► **Theorem 1.** *There exists a distributed algorithm for computing a Hamiltonian cycle in DIRAC graphs that runs in  $O(\log n)$  rounds in the CONGEST model. The algorithm succeeds with high probability.<sup>4</sup>*

We also prove the following, more general, theorem in the full version.

► **Theorem 2.** *There exists a distributed algorithm for computing a Hamiltonian cycle in ORE graphs and a Hamiltonian path in RK graphs that runs in  $O(\log n)$  rounds in the CONGEST model. The algorithm succeeds with high probability.*

A nice outcome of Theorem 2 is that a sequential simulation of the CONGEST algorithm for the RK graphs yields a polynomial time sequential algorithm for finding a Hamiltonian path in these graphs as well.

## 1.1 High-level Description of the Algorithm

Our algorithm maintains a path-cover of the graph, where a path-cover is a set of paths in the graph such that each vertex of the graph belongs to exactly one of the paths in the set.

Initially, the path-cover consists of paths of constant length. Hence the size of the initial path-cover is linear in  $n$ . Then the algorithm proceeds in iterations, where in each iteration, the size of the path-cover decreases by a constant factor, with constant probability. Consequently, after  $\Theta(\log n)$  iteration, the size of the path-cover is 1. Namely, a Hamiltonian path is found. The decrease in the size of the path-cover occurs as in each iteration (with constant probability) a constant fraction of the paths are merged into other paths, as we describe next.

We consider three types of merges. An *elementary merge* occurs when a path  $P$  is merged into a path  $Q$  by connecting the endpoints of  $P$  to the endpoints of an edge of  $Q$ . A *concatenation merge* occurs when two paths are merged by connecting their endpoints. Finally, a *cycle merge* occurs when merging two cycles that are connected with an edge into a single path.

In each iteration, the algorithm proceeds as follows. First, the paths of the path-cover are paired. Now, two phases are performed, as follows. In the first phase of each iteration, only pairs of paths for which a special condition (which we describe momentarily) holds are merged. The special condition guarantees that each one of these pairs can be merged into a single path. Specifically, a pair of paths,  $(P, Q)$  satisfy this special condition if the subgraph induced on the vertices of each one of them has a Hamiltonian cycle and additionally  $P$  and  $Q$  are connected with an edge to each other.

In the second phase of each iteration, concatenations and elementary merges are performed. The merges that are performed in this phase are selected as follows. Let  $P$  be a path in the path cover. The *elements* of  $P$  consists of its edges and its endpoints. Initially, each element

<sup>4</sup> We say that an event occurs *with high probability (w.h.p.)* if it occurs with probability at least  $1 - 1/\text{poly}(n)$ .

of  $P$  reserves itself to at most a single path,  $Q$ . The path  $Q$  is selected as follows. If the (reserved) element is an endpoint of  $P$ ,  $v$ , then  $Q$  is a path with an endpoint incident to  $v$ , chosen uniformly at random from the set of such paths. Otherwise, if the (reserved) element is an edge <sup>5</sup> of  $P$ ,  $(u, v)$ , then  $Q$  is a path with an endpoint incident to  $u$ , chosen uniformly at random from the set of such paths. A reservation of an element of  $P$  to a path  $Q$  grants  $Q$  the exclusive right to be merged into  $P$  using this element. The purpose of these reservations is to avoid a scenario in which two different paths are trying to merge into another path by using the same element.

We say that a reservation of an element is *useful* for a path  $Q$  if  $Q$  can be merged into another path via this element. By construction, all reservations of endpoints are useful. However, a reservation of an edge may be non-useful since the decision to reserve an edge to a path,  $Q$ , is done only by one of the endpoints of  $e$ . Hence, it might be the case that the other endpoint of  $e$  is not adjacent to the other endpoint of  $Q$ . Nonetheless, we show that on expectation, a constant fraction of the paths will have at least one useful reservation. After setting the reservations, each path is notified of the set of its useful reservations (if any). This can be carried out without congestion because of the exclusivity of the reservations. Then each path arbitrarily selects one of its useful reservations.

At this point, each path that has a useful reservation can be merged into another path via the respective reserved element exclusively. However, there are two problems with executing these merges. The first problem is that we want to avoid lengthy sequences of merges as this blows up the round complexity of the algorithm. Roughly speaking, this comes from the fact that when we merge paths (possibly many) into a single new path, all vertices of the corresponding (old) paths are updated about the identity of the new path. Moreover, for each new path (which is an outcome of possibly many merging operations), the algorithm constructs a spanning tree, of depth 2, which spans the vertices of the path. See more details in the full version on the role of these spanning trees in our algorithm.

The second problem is that these sequences of merges may be conflicting. A simple example of a conflict is when a graph  $P$  tries to merge into  $Q$  via an edge of  $Q$ , and  $Q$  tries to merge into  $P$  via an edge of  $P$ . Clearly, these two merges cannot be carried out simultaneously. Moreover, this example can be extended into arbitrarily long cycles.

Fortunately, these two problems can be remedied by the following simple idea. Each path tosses a fair coin. Then each path,  $P$ , is merged via its selected (reserved) element only if  $P$  tossed **tails** and  $Q$  tossed **heads**, where  $Q$  is the path of the respective element. On expectation  $1/4$  of the merges will be in the “right” orientation. In our analysis, we prove that this suffices for our needs (see more details on Subsection 4.3). This completes the description of the second phase of each iteration and concludes the description of the algorithm.

## 1.2 Correctness and Analysis of the Algorithm

The analysis of the algorithm has two ingredients. The first and main ingredient is showing that for any fixed iteration, the size of the path cover decreases by a constant fraction on expectation. The second ingredient is showing that after  $\Theta(\log n)$  iterations, with high probability, all paths are merged into a single Hamiltonian path.

---

<sup>5</sup> We note that although the graph is undirected, we keep an orientation on the edges of the paths of the path cover (so the obtained paths are directed).

For the first ingredient of the analysis, we define the notion of being a *good path*. Roughly speaking, a path is good if there are sufficiently many edges connecting its endpoints to vertices of other paths (the actual definition is more cumbersome than this, but this is essentially the property that we need). This property guarantees that any good path has many options for merging into other paths. In particular, the number of options is linear in the size of the current path cover (Lemma 14). We use this fact to show that, on any fixed iteration, a good path receives a useful reservation with constant probability (Claim 15).

Finally, we prove that there are sufficiently many good paths. Recall that at the beginning of each iteration, the paths are paired. In the algorithm analysis, we prove that for each one of the pairs of paths,  $(P, Q)$ , that were not merged in the first phase, either  $P$  or  $Q$  are good with respect to the current path-cover. We then show that consequently, this guarantees that with constant probability, a constant fraction of these paths will be merged into other paths in the second phase of the algorithm.

### 1.3 Adaptation of the Algorithm for ORE and RK Graphs

Since the family of RK graphs contains ORE graphs we, from now on, focus on RK graphs. We begin by proving some structural properties of RK graphs. One of these properties is that the vertices in RK graphs can be partitioned into 3 sets  $A$ ,  $C$ , and  $H$  where all the vertices in  $H$  satisfy Dirac's condition and the subgraph induced on each one of the sets  $A$  and  $C$ , form a clique. This structure allows us to perform as in the algorithm for DIRAC graphs with the only difference that at the beginning of each iteration, we get rid of (almost) all the paths in which one endpoint is not in  $H$ . Another technicality that we need to handle is the fact that our algorithm for DIRAC graphs uses spanning trees to manage the communication within the different paths in the path cover. In RK graphs these spanning trees may not span the entire respective paths. However, we show that with a slight adaptation, it is possible to maintain communication within the paths while adding only a constant factor blow-up in the round complexity.

### 1.4 Related Work

#### Parallel Algorithms

As mentioned above, Dahlhaus et al. [10] gave a  $O(\log^4 n)$  CREW-PRAM algorithm that uses a linear number of processors to find a Hamiltonian cycle in Dirac graphs. Another generalization of Dirac graphs are *Chvátal graphs*. A graph is called a Chvátal graph if its degree sequence  $d_1 \leq d_2 \leq \dots \leq d_n$  satisfies that for every  $k < n/2$ ,  $d_k \leq k$  implies that  $d_{n-k} \geq n - k$ . Chvátal proved (see e.g., [3]) that Chvátal graphs are also Hamiltonian. In [6] a sequential polynomial time algorithm that finds Hamiltonian cycles in Chvátal graphs has been shown. Sárközy [23] proved that a deterministic  $O(\log^4 n)$  time EREW-PRAM algorithm with a polynomial number of processors that finds a Hamiltonian cycle in a  $\eta$ -Chvátal graphs exists. A graph is called  $\eta$ -Chvátal graphs if for every  $k < n/2$ ,  $d_k \leq \min\{k + \eta n, n/2\}$ , it holds that  $d_{n-k-\eta n} \geq n - k$ , where  $0 < \eta < 1$ .

#### Distributed Algorithms with a Promise

It is known that a random  $G(n, p)$  graph contains w.h.p. a Hamiltonian cycle if  $p$  is at least  $(\log n + \log \log n + t(n))/n$ , for any divergent function  $t(n)$  [4]. Thus for any such  $p$ , the problem of deciding whether the graph is Hamiltonian becomes trivial; however, finding the Hamiltonian path or cycle, in this case, is still non-trivial. The problem of

finding a Hamiltonian cycle in the distributed setting was initially studied by Levy et al. [16] that provided an upper bound whose round complexity is  $O(n^{3/4+\epsilon})$  that w.h.p. finds a Hamiltonian cycle given that  $p = \omega(\sqrt{\log n}/n^{1/4})$ . Thereafter, other upper bounds were designed in the CONGEST model (in which the message size is bounded) for dealing with various ranges of  $p$ , as described next. The algorithm of Chatterjee et al. [8] works for  $p \leq \frac{c \ln n}{n^\delta}$ , ( $0 < \delta \leq 1$ ) and finds a Hamiltonian cycle w.h.p. in  $\tilde{O}(n^\delta)$  rounds. For  $p$  in  $\tilde{\Omega}(1/\sqrt{n})$ , Turau [24], provides an algorithm that finds w.h.p. a Hamiltonian cycle in  $O(\log n)$  rounds. More recently, Ghaffari and Li [13] showed the existence of a distributed algorithm for finding a Hamiltonian cycle, w.h.p., in  $G(n, d)$  for  $d = C \log n$  whose round complexity is  $2^{O(\sqrt{\log n})}$ , where  $C$  is a sufficiently large constant.

### Lower Bounds in the CONGEST Model

It is well known that certain properties can not be decided in the CONGEST model in a number of rounds which is  $o(n^2)$  [21]. As mentioned above, Bachrach et al. [1] proved a lower bound of  $\tilde{\Omega}(n^2)$  rounds for various problems in the CONGEST model, including Testing Hamiltonicity in general graphs.

## 1.5 Comparison with the Algorithm of Dahlhaus et al. [10]

As mentioned above, our algorithm builds on ideas from the algorithm of Dahlhaus et al. [10] for finding a Hamiltonian path in Dirac's graphs. Their algorithm also proceeds in iterations such that at each iteration it first performs cycle merges, then it performs concatenation merges and finally, it performs elementary merges. However, the specific structure of their algorithm and how these merges are selected are quite different from our algorithm.

We shall demonstrate several structural differences without going into all the details of their algorithm (which is more involved than our algorithm). These differences serve us in obtaining an improved round complexity and a simpler algorithm. Thereafter we shall emphasize the specific differences that arise from the fact that our algorithm works in the CONGEST model rather than the PRAM model.

The first structural difference is that on each iteration, before their algorithm turns into performing elementary merges it first has to exhaust most of the cycle merges, which requires an inner loop of  $\Theta(\log n)$  steps and thereafter it exhausts most of the concatenation merges by executing a special subroutine of Israeli and Shiloach [14] which returns both a vertex cover and an approximated maximum matching.

This subroutine is executed twice. One time on the subgraph induced on the endpoints of the paths in the path cover and another time on an auxiliary graph where on one side we have the set of paths and on the other side we have the set of edges composing the paths.

The reason that their algorithm exhausts the three types of merges in phases is that the progress of each phase relies on the exhaustion of the merges of the previous step.

For comparison, per iteration, our algorithm performs only one step of cycle merges and then one step in which both concatenation and elementary merges are performed simultaneously. We prove that this is sufficient to make enough progress per iteration.

Another difference is that in their algorithm, at the beginning of each iteration, every path is classified into one of two types. We are able to avoid this classification altogether and use a somewhat different classification only in the analysis.

We next list several challenges that arise specifically in the CONGEST model and in particular do not allow us to easily translate the algorithm of [10] into the CONGEST model.

The first problem is that we do not have shared memory among the processors so how can we efficiently merge even a pair of paths? To this end, our algorithm maintains spanning trees, of depth 2, on each one of the paths in the constructed path cover. Therefore, initially, we have a linear (in  $n$ ) number of spanning trees (and this number decreases as the algorithm progresses). We show that each edge participates in at most 2 different spanning trees and so communication within vertices of the same path can be maintained without causing congestion.

Another problem is concerned with elementary merges. Consider a path cover  $\mathcal{P}$  and an edge  $(u, v)$  on a path  $P \in \mathcal{P}$ . An elementary merge of a path  $Q$  into  $P$  can be performed via  $(u, v)$  only if one endpoint of  $Q$  is adjacent to  $u$  and the other is adjacent to  $v$ . However, there might be many endpoints that are adjacent to  $u$  or  $v$ , so how can we find the set of paths that can be merged via  $(u, v)$  into  $P$  without communicating too much between  $u$  and  $v$ ? As mentioned above, we show that we don't have to find this set. Specifically, we show that when  $u$  reserves the edge  $(u, v)$  to an endpoint that is adjacent to  $u$ , picked uniformly at random, then every path receives a useful reservation with constant probability. We remark that in this case, we rely on the randomness of our algorithm while the algorithm of [10] is deterministic.

Finally, we want to avoid long sequences of merges so we won't have to deal with long sequences of updates. To this end, we use the coin tosses of the vertices and perform merge operations only if they agree with the orientation defined by the coin tosses. As described above, this is also useful for avoiding conflicting merges. Consequently, the merges can be carried out simultaneously with very little and local coordination.

## 2 Paths-Merge Types and Paths Classification

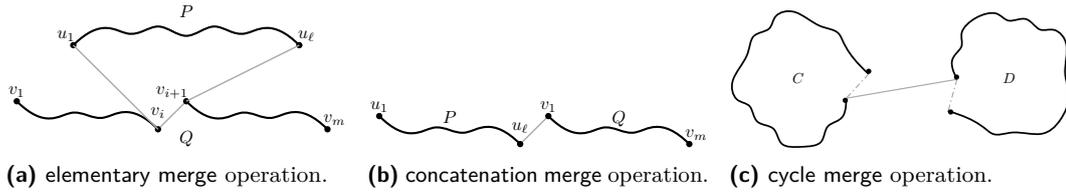
### Notation

Let  $G = (V, E)$  be an undirected simple graph, where  $V$  is the vertex set, and  $E \subseteq \{\{u, v\} \mid u, v \in V\}$  is the edge set. Let  $n$  denote  $|V|$  and let  $m$  denote  $|E|$ . For a path  $P = (u_1, \dots, u_\ell)$ , we denote by  $V(P)$  the vertex set of  $P$ , i.e.,  $V(P) \triangleq \{u_1, \dots, u_\ell\}$ . For  $v \in V$ , let  $N(v)$  denote the neighbors set of  $v$  in  $G$ , that is  $N(v) = \{u \in V \mid \{v, u\} \in E\}$ . Let  $d(v)$  denote the degree of  $v$ , i.e.,  $d(v) = |N(v)|$ . For a pair of vertices  $u$  and  $v$ , let  $\delta(u, v)$  denote the length of a shortest path between  $u$  and  $v$ . We say that a set of paths in  $G$ ,  $\{P_i\}_{i=0}^{k-1}$ , is a *path-cover* of  $G$  if its union covers the vertex set of  $G$ , that is, if  $\bigcup_{i=0}^{k-1} V(P_i) = V(G)$ . For a path  $P$  in  $G$ , let  $d_P(v)$  denote the number of neighbours of  $v$  in  $P$ . For a path  $P = (u, \dots, v)$ , we refer to the vertices  $u$  and  $v$  as the *endpoints* of  $P$ .

### Path Merging Types

Through the course of the algorithm's execution, the algorithm performs three kinds of path merging depending on the paths at hand: an elementary merge, a concatenation, and cycle merging (see Figure 1). These merge operations are defined as follows.

► **Definition 3** (Elementary merge [10]). *Let  $P = (u_1, \dots, u_\ell)$  and  $Q = (v_1, \dots, v_m)$  be two disjoint paths. If  $\{u_1, v_i\}, \{u_\ell, v_{i+1}\} \in E$ , then  $(v_1, \dots, v_i, u_1, \dots, u_\ell, v_{i+1}, \dots, v_m)$  is a path. If  $\{u_1, v_{i+1}\}, \{u_\ell, v_i\} \in E$ , then  $(v_1, \dots, v_i, u_\ell, \dots, u_1, v_{i+1}, \dots, v_m)$  is a path. In either case, we say that we merged  $P$  into  $Q$  along the edge  $\{v_i, v_{i+1}\}$ . We call this step an elementary merging operation.*



■ **Figure 1** Path merging types.

► **Definition 4** (Concatenation [10]). Let  $P = (u_1, \dots, u_\ell)$  and  $Q = (v_1, \dots, v_m)$  be two disjoint paths. If there is an edge connecting an endpoint of  $P$  (either  $u_1$ , or  $u_\ell$ ) and an endpoint  $v \in \{v_1, v_m\}$  of  $Q$ , then we can use any of these edges to concatenate  $P$  and  $Q$  and say that we concatenated  $P$  to  $Q$  along vertex  $v$ . We call this operation a concatenation.

► **Definition 5** (Cycle Merging [10]). Let  $C$  and  $D$  be two disjoint cycles. If there is an edge connecting a vertex from  $C$  and a vertex from  $D$ , we can use this edge to get a path that passes through all the vertices of  $C$  and  $D$ . We call this operation a cycle merging.

### Paths Classification

For the sake of the analysis of the algorithm we classify the paths that the algorithm maintains as *sociable paths* (à la Dahlhaus et al. [10]) or as *cycled paths*, as follows.

► **Definition 6** ([10]). Let  $P = (u, \dots, v)$  be a path in a graph  $G$ . We say that the path  $P$  is sociable if  $d_P(u) + d_P(v) + 1 \leq |V(P)|$ .

► **Definition 7** (Cycled Path). Let  $P = (u_1, \dots, u_\ell)$  be a path in a graph  $G$ . We say that the path  $P$  is cycled if  $\{u_1, u_\ell\} \in E$  or if there exists an edge of  $P$ ,  $\{u_i, u_{i+1}\}$  such that both  $\{u_1, u_{i+1}\}, \{u_i, u_\ell\} \in E$ .

It is easy to see that if a path  $P$  is cycled, then the subgraph induced on  $V(P)$  is Hamiltonian. We shall use the following basic claim, the proof of which is deferred to the full version.

▷ **Claim 8.** Let  $P = (u_1, \dots, u_\ell)$  be a path that is not cycled. Then  $P$  is sociable.

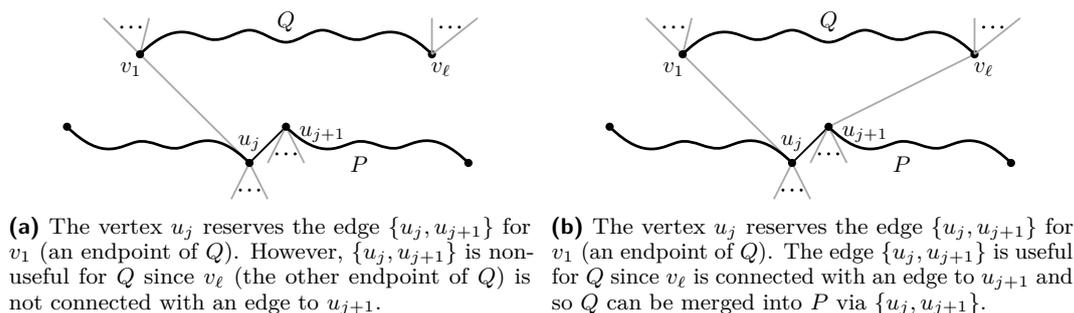
## 3 The Algorithm

In this section, we list our distributed algorithm (see Algorithm 1) without giving all the details of implementation. We then prove its correctness in Section 4 and in the full version we discuss in more detail how the algorithm is implemented in the CONGEST model. We establish the following theorem.

► **Theorem 1.** *There exists a distributed algorithm for computing a Hamiltonian cycle in DIRAC graphs that runs in  $O(\log n)$  rounds in the CONGEST model. The algorithm succeeds with high probability.*

### 3.1 Listing of the Distributed Algorithm

Our algorithm begins with finding an initial path-cover of the graph in which each path is of length at least 2, denoted by  $\mathcal{P}_0$ . Then the algorithm proceeds in  $\Theta(\log n)$  iterations (in which the size of the path-cover decreases by a constant fraction with constant probability).



■ **Figure 2** Useful versus non-useful reservations.

We denote the path-cover at the beginning of the  $i$ -th iteration by  $\mathcal{P}_i$ . At the beginning of each iteration, the paths in  $\mathcal{P}_i$  are partitioned into pairs. Each pair of paths  $(P, Q)$  such that  $P$  and  $Q$  are connected with an edge, and both  $P$  and  $Q$  are cycled are merged (Step 5). We denote by  $\mathcal{P}_i^a$  the resulting path-cover.

Thereafter, each edge and endpoint of a path in  $\mathcal{P}_i^a$  reserves itself to an endpoint of another path in  $\mathcal{P}_i^a$ , which is picked uniformly at random.

Then, each path  $P$  in the path-cover selects out of the elements that were reserved to it (i.e., either an edge or an endpoint) a single element,  $\ell$ , such that  $P$  can be merged to another path via  $\ell$  (we refer such elements as *useful* for  $P$ ).

Each path,  $P$ , tosses a random fair coin,  $y_P$ . Next, all the merges are performed simultaneously where a path  $P$  is merged via its selected element,  $\ell$ , to a path  $Q$  only if  $y_P = \text{tails}$  and  $y_Q = \text{heads}$ .

► **Remark 9.** When a path  $P$  is merged to a path  $Q$  where  $y_P = \text{tails}$  and  $y_Q = \text{heads}$ , the orientation of  $Q$  is kept, and the orientation of  $P$  may be reversed to maintain consistency with the orientation of  $Q$ . For example, if  $P = (y_1, \dots, y_\ell)$  is merged to  $Q$  via  $\{u, v\}$  where the orientation of  $\{u, v\}$  is from  $u$  to  $v$  and  $y_1$  is connected to  $v$  and  $y_\ell$  is connected to  $u$  then the orientation of  $P$  is reversed after the merge.

## 4 Correctness of the Algorithm

In this section, we prove the correctness of our algorithm.

We begin by giving a lower bound on the number of possible merges for each path in the path-cover. We then use this bound to give a lower bound on the expected number of merges carried out in each iteration. Finally, we prove that with high probability after  $\Theta(\log n)$  iterations, all paths are merged into a single path. Some of the proofs are deferred to the full version.

### 4.1 Number of Possible Merges for Good Paths

In this section, we provide the proof of Lemma 11 which gives a lower bound on the number of possible ways a path  $P$  can be merged into a path  $Q$ . We then define the notion of *good* paths (Definition 13). Roughly speaking, a path is good if its endpoints are neighbors of sufficiently many vertices that belong to other paths in the path cover. Thereafter, we use Lemma 11 to give a lower bound on the total number of ways a good path,  $P$ , can be merged into any other path in the path-cover (Lemma 14).

■ **Algorithm 1** Finding a Hamiltonian path in a Dirac graph.

---

**Input:** A Dirac graph  $G = (V, E)$ .  
**Invariant:** The algorithm maintains a path-cover  $\mathcal{P}_i$  for every  $i \geq 0$ . The path-cover  $\mathcal{P}_{i+1}$  is computed from  $\mathcal{P}_i$  via elementary merges, cycle merges and concatenation merges operations.  
**Output:**  $\mathcal{P}_{\Theta(\log n)}$  is a Hamiltonian path. w.h.p. // For the exact constant within the  $\Theta$  notation we refer the reader to Lemma 18.

---

- 1 Compute a path-cover,  $\mathcal{P}_0$  in the graph such that each path is of size at least 2.
- 2 **for**  $i \leftarrow 0$  **to**  $\ell = \Theta(\log n)$  **do**
- 3     Pair all paths (except for at most one) and let  $\mathcal{I}_i$  denote the set of these pairs of paths.
- 4     Let  $\mathcal{L}_i \subseteq \mathcal{I}_i$  denote the set of paired paths that have an edge between them and for which both paths are cycled.
- 5     Perform cycle merges on all pairs in  $\mathcal{L}_i$ . // see Definition 5
- 6     Let  $\mathcal{P}_i^a$  denote the current path-cover
- 7     For every  $v \in V$  let  $D_i(v)$  denote the subset of endpoints of paths in  $\mathcal{P}_i^a$ ,  $u$ , such that  $\{u, v\} \in E$
- 8     **for every**  $P \in \mathcal{P}_i^a$  where  $P = (u_1, \dots, u_\ell)$  **do**
- 9         Every  $u_j$  for  $j \in \{1, \dots, \ell - 1\}$  picks an endpoint  $v$  u.a.r. from  $D_i(u_j)$  and reserves the edge  $(u_j, u_{j+1})$  for  $v$ .
- 10         Additionally, each endpoint of  $P$ ,  $v$ , reserves itself to an endpoint which is picked u.a.r. from  $D_i(v)$ .
- 11          $y_P \leftarrow \begin{cases} \text{heads,} & \text{w.p. } 1/2, \\ \text{tails,} & \text{o.w.} \end{cases}$ .
- 12          $M_i \leftarrow \emptyset$
- 13         **for every**  $P \in \mathcal{P}_i^a$  **do**
- 14             We say that an endpoint or an edge,  $x$ , is useful for  $P$  (w.r.t.  $\mathcal{P}_i^a$ ) if  $P$  can be concatenated or merged along  $x$  to another path in  $\mathcal{P}_i^a$ .
- 15             Let  $S_i(P)$  denote the set of elements reserved for the endpoints of  $P$  in Steps 8-10 which are also *useful* for  $P$ .
- 16              $P$  picks arbitrarily one of the elements in  $S_i(P)$  (assuming it is not empty) and adds it to  $M_i$ .
- 17     Perform concatenation merges and elementary merges with accordance to  $M_i$  and the  $y_P$  variables: a path  $P_1$  merges into a path  $P_2$  if there is a corresponding merge operation in  $M_i$  and if  $y_{P_1} = \text{tails}$  and  $y_{P_2} = \text{heads}$ . // see Definitions 3, 4
- 18 **return**  $\mathcal{P}_\ell$ .

---

Let  $P = (u, \dots, v)$  and  $Q = (a, \dots, b)$  be disjoint paths in  $G$ . Let  $M(P, Q)$  denote the number of edges along which one can merge  $P$  into  $Q$  via elementary merging plus the number of vertices along which one can concatenate  $P$  to  $Q$  (see Definitions 3 and 4, respectively). Let  $d(P, Q)$  denote the number of endpoints of  $Q$  connected with an edge to an endpoint of  $P$ .

We begin with extending the following lemma from [10].

► **Lemma 10** (Lemma. 5.2.5 [10], restated). *Let  $P = (u, \dots, v)$  and  $Q$  be disjoint paths in  $G$ . If  $d(P, Q) = 0$ , then the number of edges along which one can merge  $P$  into  $Q$  via elementary merging operations is at least  $d_Q(u) + d_Q(v) - |V(Q)| + 1$ .*

To achieve better round complexity, our algorithm performs concatenation and elementary merges simultaneously. To this end, we prove the following lemma, which extends Lemma 10 so that it also applies to cases where  $d(P, Q) > 0$ .

► **Lemma 11.** *Let  $P = (u, \dots, v)$  and  $Q = (a, \dots, b)$  be disjoint paths in  $G$  such that  $|V(P)|, |V(Q)| \geq 2$ . Then,*

$$M(P, Q) \geq d_Q(u) + d_Q(v) - |V(Q)| + 1. \quad (1)$$

We next give a lower bound on the number of merging operations which applies for a subset of paths in the path cover. We shall use the following definition.

► **Definition 12.** *Let  $\mathcal{P}$  be a path cover and let  $A(\mathcal{P})$  denote the set of paths,  $P = (u, \dots, v)$ , in  $\mathcal{P}$  such that there exists a path  $P' \in \mathcal{P}$  for which  $|V(P')| \geq |V(P)|$  and  $d_{P'}(u) + d_{P'}(v) = 0$ .*

In words,  $A(\mathcal{P})$  is the set of paths,  $P \in \mathcal{P}$  for which there exists another path,  $P' \in \mathcal{P}$ , which is not shorter than  $P$  and for which the endpoints of  $P$  are not adjacent to any of the vertices composing  $P'$ . We next define the notion of *good path*.

► **Definition 13 (good path).** *Let  $\mathcal{P}$  be a path cover. A path  $P \in \mathcal{P}$  is good (w.r.t.  $\mathcal{P}$ ) if it is sociable or in  $A(\mathcal{P})$ .*

The following lemma gives a lower bound on the number of merging operations for any path which is good. In the proof of the lemma, we extend ideas from Corollary 5.2.6 and Lemma 5.2.7 of Dahlhaus et al. [10]. Our extension allows us to drop the stringent requirement of Dahlhaus et al. [10] that the endpoints of  $P$  are not adjacent to endpoints of any other path in the path cover. This allows us to support the execution of both concatenations and elementary merging operations simultaneously.

► **Lemma 14.** *Let  $\mathcal{P}$  be a path-cover of  $G$ . For  $P \in \mathcal{P}$  define  $M(P) \stackrel{\text{def}}{=} \sum_{Q \in \mathcal{P} \setminus \{P\}} M(P, Q)$ . Then, for every good path,  $P \in \mathcal{P}$  it holds that  $M(P) \geq |\mathcal{P}|$ .*

## 4.2 Expected Number of Merges

In this section, we prove Claim 16 which states that the expected number of merges of each iteration is sufficiently large.

More specifically, for a fixed iteration  $i$ , we prove that if the number of cycle-merges performed at Step 5 is below some threshold (specifically if  $|\mathcal{L}_i| \leq |\mathcal{P}_i|/12$ ), then the expected size of  $M_i$  (the set of concatenation merges and elementary merges added in Step 16) is a constant fraction of the size of  $\mathcal{P}_i$ . We first prove the following claim which gives a lower bound on the probability that a good path receives a useful reservation.

▷ **Claim 15.** Fix an iteration  $i$ . For any  $P$  which is good with respect to  $A(\mathcal{P}_i^a)$ , it holds that  $P$  receives a useful reservation for at least one element with probability at least  $1/3$ .

*Proof.* Fix an iteration  $i$  and let  $P$  be a good path with respect to  $A(\mathcal{P}_i^a)$ . Let  $F$  denote the set of edges and endpoints that  $P$  can be merged to (see Definitions 3, 4) in  $\mathcal{P}_i^a$ . By Lemma 14, it holds that  $|F| \geq |\mathcal{P}_i^a|$ . Let  $x = 2|\mathcal{P}_i^a|$  denote the total number of endpoints of paths in  $\mathcal{P}_i^a$ . Since every edge and endpoint in  $F$  is reserved for  $P$  with a probability of at least  $1/x$ , the probability that  $P$  received at least one reservation of an element in  $F$  is at least  $1 - (1 - 1/x)^{|F|}$ . since  $(1 - 1/x)^{|F|} \leq (1 - 1/(2|\mathcal{P}_i^a|))^{|\mathcal{P}_i^a|} \leq 1/\sqrt{e}$ , it holds that this probability is at least  $1/3$ . ◁

▷ **Claim 16.** If  $\mathcal{L}_i \leq |\mathcal{P}_i|/12$  then  $\mathbb{E}(|M_i| \mid \mathcal{P}_i) \geq |\mathcal{P}_i|/36$ .

*Proof.* We first observe that in every pair  $(P, Q) \in \mathcal{I}_i \setminus \mathcal{L}_i$  at least one path is good with respect to  $\mathcal{P}_i^a$ . To see this, observe that there are two cases. The first case is that  $P$  and  $Q$  are not connected with an edge. This implies that at least one of them is in  $A(\mathcal{P}_i^a)$ . The second case is that at least one of them is not cycled, which by Claim 8 implies that at least one of them is sociable, as desired. We next lower bound the number of pairs in  $\mathcal{I}_i \setminus \mathcal{L}_i$ . Since the number of pairs is at least  $(|\mathcal{P}_i| - 1)/2 \geq |\mathcal{P}_i|/4$  and  $|\mathcal{L}_i| < |\mathcal{P}_i|/12$  it holds that  $|\mathcal{I}_i \setminus \mathcal{L}_i| \geq |\mathcal{P}_i|/4 - |\mathcal{P}_i|/12 = |\mathcal{P}_i|/6$ .

Therefore, by Claim 15, the expected size of merges that are added to  $M_i$  is at least  $|\mathcal{I}_i \setminus \mathcal{L}_i|/2 \cdot (1/3) \geq |\mathcal{P}_i|/36$ , as required. ◀

### 4.3 The Progress of Each Iteration

In this section, we prove the following lemma.

► **Lemma 17.** *For any iteration  $i$  of Algorithm 1 it holds that  $\mathbb{E}(|\mathcal{P}_{i+1}|) \leq (1 - 1/144) \cdot \mathbb{E}(|\mathcal{P}_i|)$ .*

*Proof.* Fix an iteration  $i$ . If  $\mathcal{L}_i \geq |\mathcal{P}_i|/12$  then at least  $|\mathcal{P}_i|/12$  paths of  $\mathcal{P}_i$  are merged and so  $|\mathcal{P}_{i+1}| \leq (1 - 1/12)|\mathcal{P}_i|$ , as desired.

Otherwise, by Claim 16,  $\mathbb{E}(|M_i|) \geq |\mathcal{P}_i|/36$ . Consider a merge operation in  $M_i$  in which path  $P$  is merged into path  $Q$ . This merge is carried in Step 17 only if  $y_P = \mathbf{tails}$  and  $y_Q = \mathbf{heads}$ , which happens with probability  $1/4$ . We denote these merge operations by  $M'_i$ , hence  $\mathbb{E}(|M'_i|) \geq |\mathcal{P}_i|/(36 \cdot 4) = |\mathcal{P}_i|/144$ . Moreover, since  $|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i| - |M'_i|$  (recall that merges can occur before Step 17), it follows that  $\mathbb{E}(|\mathcal{P}_{i+1}| \mid \mathcal{P}_i) \leq |\mathcal{P}_i| - \mathbb{E}(|M'_i| \mid \mathcal{P}_i) \leq (1 - 1/144) \cdot |\mathcal{P}_i|$ .

The lemma follows since

$$\mathbb{E}(|\mathcal{P}_{i+1}|) = \sum_{\mathcal{P}_i} \Pr(\mathcal{P}_i) \cdot \mathbb{E}(|M'_i| \mid \mathcal{P}_i) \leq \sum_{\mathcal{P}_i} \Pr(\mathcal{P}_i) \cdot (1 - 1/144) \cdot |\mathcal{P}_i| = (1 - 1/144) \cdot \mathbb{E}(|\mathcal{P}_i|),$$

as required. ◀

#### 4.3.1 Number of Iterations

In this section, we prove that if the for-loop in Step 2 of Algorithm 1 performs  $\Theta(\log n)$  iterations, then the path that is returned at the end of the algorithm is Hamiltonian w.h.p.

We note that, although the algorithm uses independent coin tosses between different iterations, the success of two different iterations are random variables that may be dependent. Therefore we cannot use concentration bounds that assume the independence of the random variables (such as Chernoff's bound). Roughly speaking, the dependence comes from the fact that the constructed path cover depends on the coin tosses of previous iterations. Nonetheless, we can show that  $\Theta(\log n)$  iterations are sufficient.

The proof of our main theorem (Theorem 1) follows directly from the following lemma.

► **Lemma 18.** *The path returned in Step 18 is Hamiltonian w.h.p.*

*Proof.* Lemma 17 and the fact that  $|\mathcal{P}_0| \leq n$  imply that

$$\mathbb{E}(|\mathcal{P}_\ell|) \leq (1 - 1/144)^\ell \cdot \mathbb{E}(|\mathcal{P}_0|) \leq (1 - 1/144)^\ell \cdot n.$$

It follows that for  $\ell \geq \frac{2}{\log_2(144/143)} \cdot \log_2 n$ , it holds that  $\mathbb{E}(|\mathcal{P}_\ell|) \leq \frac{1}{n}$ .

Thus, by Markov's Inequality, it follows that the probability that the Algorithm fails is

$$\Pr(|\mathcal{P}_\ell| > 1) \leq \frac{\mathbb{E}(|\mathcal{P}_\ell|)}{1} \leq \frac{1}{n},$$

as required. ◀

---

## References

- 1 Nir Bachrach, Keren Censor-Hillel, Michal Dory, Yuval Efron, Dean Leitersdorf, and Ami Paz. Hardness of distributed optimization. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 238–247. ACM, 2019. doi:10.1145/3293611.3331597.
- 2 Jean-Claude Bermond. On hamiltonian walks. In *5th British Combinatorial Conference, 1975, Congressus Numerantium 15, Utilitas Math Pub.*, pages 41–51, 1976.
- 3 Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.
- 4 Béla Bollobás. *Random Graphs, Second Edition*, volume 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2011. doi:10.1017/CB09780511814068.
- 5 J Adrian Bondy. Large cycles in graphs. *Discrete Mathematics*, 1(2):121–132, 1971.
- 6 J Adrian Bondy and Vasek Chvátal. A method in graph theory. *Discrete Mathematics*, 15(2):111–135, 1976.
- 7 John Adrian Bondy. *Longest paths and cycles in graphs of high degree*. Department of Combinatorics and Optimization, University of Waterloo, 1980.
- 8 Soumyottam Chatterjee, Reza Fathi, Gopal Pandurangan, and Nguyen Dinh Pham. Fast and efficient distributed computation of hamiltonian cycles in random graphs. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 764–774. IEEE, 2018.
- 9 E Dahlhaus, P Hajnal, and M Karpinski. Optimal parallel algorithm for the hamiltonian cycle problem on dense graphs. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 186–193. IEEE, 1988.
- 10 Elias Dahlhaus, Peter Hajnal, and Marek Karpinski. On the parallel complexity of hamiltonian cycle and matching problem on dense graphs. *Journal of Algorithms*, 15(3):367–384, 1993.
- 11 Evelyne Flandrin, HA Jung, and Hao Li. Hamiltonism, degree sum and neighborhood intersections. *Discrete mathematics*, 90(1):41–52, 1991.
- 12 Irène Fournier and Pierre Fraisse. On a conjecture of bondy. *Journal of Combinatorial Theory, Series B*, 39(1):17–26, 1985.
- 13 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *32nd International Symposium on Distributed Computing*, 2018.
- 14 Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):57–60, 1986.
- 15 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 16 Eythan Levy, Guy Louchard, and Jordi Petit. A distributed algorithm to find hamiltonian cycles in  $G(n, p)$  random graphs. In Alejandro López-Ortiz and Angèle M. Hamel, editors, *Combinatorial and Algorithmic Aspects of Networking*, pages 63–74, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 17 Hao Li. Generalizations of dirac's theorem in hamiltonian graph theory – A survey. *Discrete Mathematics*, 313(19):2034–2053, 2013. Cycles and Colourings 2011. doi:10.1016/j.disc.2012.11.025.
- 18 Nathan Linial. A lower bound for the circumference of a graph. *Discrete Mathematics*, 15(3):297–300, 1976. doi:10.1016/0012-365X(76)90031-5.
- 19 Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006.

## 19:14 Dist. CONGEST Alg. for Finding Ham. Paths in Dirac Graphs and Generalizations

- 20 Ore Oystein. Note on hamilton circuits. *The American Mathematical Monthly*, 67(1):55, January 1960. doi:10.2307/2308928.
- 21 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.
- 22 M Sohel Rahman and Mohammad Kaykobad. On hamiltonian cycles and hamiltonian paths. *Information Processing Letters*, 94(1):37–41, 2005.
- 23 Gábor N Sárközy. A fast parallel algorithm for finding hamiltonian cycles in dense graphs. *Discrete Mathematics*, 309(6):1611–1622, 2009.
- 24 Volker Turau. A distributed algorithm for finding hamiltonian cycles in random graphs in  $O(\log n)$  time. *Theoretical computer science*, 846:61–74, 2020.

# Locality Theorems in Semiring Semantics

Clotilde Bizière ✉

ENS Paris, France

Erich Grädel ✉ 

RWTH Aachen University, Germany

Matthias Naaf ✉ 

RWTH Aachen University, Germany

---

## Abstract

Semiring semantics of first-order logic generalises classical Boolean semantics by permitting truth values from a commutative semiring, which can model information such as costs or access restrictions. This raises the question to what extent classical model-theoretic properties still apply, and how this depends on the algebraic properties of the semiring.

In this paper, we study this question for the classical locality theorems due to Hanf and Gaifman. We prove that Hanf’s locality theorem generalises to all semirings with idempotent operations, but fails for many non-idempotent semirings. We then consider Gaifman normal forms and show that for formulae with free variables, Gaifman’s theorem does not generalise beyond the Boolean semiring. Also for sentences, it fails in the natural semiring and the tropical semiring. Our main result, however, is a constructive proof of the existence of Gaifman normal forms for min-max and lattice semirings. The proof implies a stronger version of Gaifman’s classical theorem in Boolean semantics: every sentence has a Gaifman normal form which does not add negations.

**2012 ACM Subject Classification** Theory of computation → Finite Model Theory

**Keywords and phrases** Semiring semantics, Locality, First-order logic

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.20

**Related Version** *Full Version:* <https://arxiv.org/abs/2303.12627> [6]

## 1 Introduction

Originally motivated by *provenance analysis in databases* (see e.g. [18, 12] for surveys), semiring semantics is based on the idea to evaluate logical statements not just by *true* or *false*, but by values in some commutative semiring  $(K, +, \cdot, 0, 1)$ . In this context, the standard semantics appears as the special case when the Boolean semiring  $\mathbb{B} = (\{\perp, \top\}, \vee, \wedge, \perp, \top)$  is used. Valuations in other semirings provide additional information, beyond truth or falsity: the tropical semiring  $\mathbb{T} = (\mathbb{R}_+^\infty, \min, +, \infty, 0)$  is used for *cost analysis*, the natural semiring  $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$  for counting evaluation strategies and proofs, and the Viterbi-semiring  $\mathbb{V} = ([0, 1]_{\mathbb{R}}, \max, \cdot, 0, 1)$  models *confidence scores*. Finite or infinite min-max semirings  $(K, \max, \min, a, b)$  can model, for instance, different *access levels* to atomic data (see e.g. [10]); valuations of a first-order sentence  $\psi$  in such *security semirings* determine the required clearance level that is necessary to access enough information to determine the truth of  $\psi$ . Further, semirings of polynomials or formal power series permit us to *track* which atomic facts are used (and how often) to establish the truth of a sentence in a given structure, and this has applications for database repairs [26] and also for the strategy analysis of games [17, 14]. Semiring semantics replaces structures by *K-interpretations*, which are functions  $\pi: \text{Lit}_A(\tau) \rightarrow K$ , mapping fully instantiated  $\tau$ -literals  $\varphi(\bar{a})$  over a universe  $A$  to values in a commutative semiring  $K$ . The value  $0 \in K$  is interpreted as *false*, while all other values in  $K$  are viewed as nuances of *true* or, perhaps more accurately, as *true, with some*



© Clotilde Bizière, Erich Grädel, and Matthias Naaf;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 20; pp. 20:1–20:15

Leibniz International Proceedings in Informatics



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

*additional information*. In provenance analysis, this is sometimes referred to as *annotated facts*. The value  $1 \in K$  is used to represent *untracked information* and is used in particular to evaluate true equalities and inequalities.

The development of semiring semantics raises the question to what extent classical techniques and results of logic extend to semiring semantics, and how this depends on the algebraic properties of the underlying semirings. Previous investigations in this direction have studied, for instance, the relationship between elementary equivalence and isomorphism for finite semiring interpretations and their definability up to isomorphism [15], Ehrenfeucht-Fraïssé games [7], and 0-1 laws [13].

The purpose of this paper is to study *locality* in semiring semantics. Locality is a fundamental property of first-order logic in classical semantics and an important limitation of its expressive power. It means that the truth of a first-order formula  $\psi(\bar{x})$  in a given structure only depends on a neighbourhood of bounded radius around  $\bar{x}$ , and on the existence of a bounded number of local substructures. Consequently, first-order logic cannot express global properties such as connectivity or acyclicity of graphs. On graphs there are natural and canonical notions of the distance between two points and of a neighbourhood of a given radius around a point. To define these notions for an arbitrary relational structure  $\mathfrak{A}$  one associates with it its *Gaifman graph*  $G(\mathfrak{A}) = (A, E)$  where two points  $a \neq b$  are adjacent if, and only if, they coexist in some atomic fact. There exist several notions of locality; the most common ones are *Hanf locality* and *Gaifman locality*, and the fundamental locality theorems for first-order logic are *Hanf's locality theorem* and *Gaifman's normal form theorem*. In a nutshell, Hanf's theorem gives a criterion for the  $m$ -equivalence (i.e. indistinguishability by sentences of quantifier rank up to  $m$ ) of two structures based on the number of local substructures of any given isomorphism type, while Gaifman's theorem states that every first-order formula is equivalent to a Boolean combination of local formulae and basic local sentences, which has many model-theoretic and algorithmic consequences. We shall present precise statements of these results in Sect. 3 and Sect. 4.

Locality thus provides powerful techniques, also for logics that go beyond first-order logic by counting properties, generalised quantifiers, or aggregate functions, [1, 21, 22, 23]. It has applications in different areas including low-complexity model-checking algorithms [19, 20], approximation schemes for logically defined optimisation problems [8], automata theory [25], computational issues on database transactions [2], and most recently also in learning theory, for the efficient learning of logical concepts [3, 5, 4]. This motivates the question, whether locality is also applicable in semiring semantics. The relevant semiring interpretations in this context are *model-defining*, which means that for any pair of complementary literals  $R\bar{a}, \neg R\bar{a}$  precisely one of the values  $\pi(R\bar{a}), \pi(\neg R\bar{a})$  is 0, and *track only positive information* which means that  $\pi(\neg R\bar{a})$  can only take the values 0 or 1. Model defining interpretations  $\pi$  define a unique structure  $\mathfrak{A}_\pi$  and we thus obtain a well-defined Gaifman graph  $G(\pi) := G(\mathfrak{A}_\pi)$ , with the associated notions of distance and neighbourhoods. The assumption that only positive information is tracked is necessary to get meaningful locality properties (see Sect. 2).

We clearly cannot generalise all known locality properties of first-order logic to semiring semantics in arbitrary commutative semirings. On semirings whose operations are not idempotent, we cannot expect a Gaifman normal form, since for computing the value of a quantified statement, we have to add or multiply values of subformulae for *all* elements of the structure, which gives an inherent source of non-locality. As a consequence, some of the locality results that we prove hold only under certain algebraic assumptions on the semiring, and further there turns out to be a difference of the locality properties of sentences and those of formulae with free variables. We shall establish the following results.

- (1) First-order formulae are Hanf-local for all semirings.
- (2) Hanf's locality theorem generalises to all fully idempotent semirings (in which both addition and multiplication are idempotent).
- (3) For formulae with free variables, Gaifman's normal form theorem does not generalise beyond the Boolean semiring.
- (4) For sentences, Gaifman's normal form theorem also fails in certain important semirings such as the natural semiring and the tropical semiring.
- (5) Over min-max semirings (and even lattice semirings), every first-order sentence has a Gaifman normal form.
- (6) In classical Boolean semantics, every sentence has a Gaifman normal form which does not introduce new negations.

The results (1), (2) on Hanf locality (Sect. 3) are proved by adaptations of the arguments for the Boolean case. The results (3) and (4) are established in Sect. 5 via specific examples of formulae that defeat locality, using simple algebraic arguments. The most ambitious result and the core of our paper is (5), a version of Gaifman's theorem for min-max semirings (Sect. 6), which we later generalise to lattice semirings (Sect. 7). It requires a careful choice of the right syntactical definitions for local sentences and, since the classical proofs in [11, 9] do not seem to generalise to semiring semantics, a new approach for the proof, based on quantifier elimination. This new approach also leads to a stronger version of Gaifman's theorem in Boolean semantics (6), which might be of independent interest.

## 2 Semiring Semantics

This section gives a brief overview on semiring semantics of first-order logic (see [16] for more details) and the relevant algebraic properties of semirings. A commutative<sup>1</sup> semiring is an algebraic structure  $(K, +, \cdot, 0, 1)$  with  $0 \neq 1$ , such that  $(K, +, 0)$  and  $(K, \cdot, 1)$  are commutative monoids,  $\cdot$  distributes over  $+$ , and  $0 \cdot a = a \cdot 0 = 0$ . We focus on semirings that are *naturally ordered*, in the sense that  $a \leq b \Leftrightarrow \exists c(a + c = b)$  is a partial order. For the study of locality properties, an important subclass are the *fully idempotent* semirings, in which both operations are idempotent (i.e.,  $a + a = a$  and  $a \cdot a = a$ ). Among these, we consider in particular all *min-max* semirings  $(K, \max, \min, 0, 1)$  induced by a total order  $(K, \leq)$  with minimal element 0 and maximal element 1, and the more general *lattice* semirings  $(K, \sqcup, \sqcap, 0, 1)$  induced by a bounded distributive lattice  $(K, \leq)$ .

For a finite relational vocabulary  $\tau$  and a finite universe  $A$ , we write  $\text{Lit}_A(\tau)$  for the set of *instantiated*  $\tau$ -literals  $R\bar{a}$  and  $\neg R\bar{a}$  with  $\bar{a} \in A^{\text{arity}(R)}$ . Given a commutative semiring  $K$ , a  $K$ -*interpretation* (of vocabulary  $\tau$  and universe  $A$ ) is a function  $\pi: \text{Lit}_A(\tau) \rightarrow K$ . It is *model-defining* if for any pair of complementary literals  $L, \neg L$  precisely one of the values  $\pi(L), \pi(\neg L)$  is 0. In this case,  $\pi$  induces a unique (Boolean)  $\tau$ -structure  $\mathfrak{A}_\pi$  with universe  $A$  such that, for every literal  $L \in \text{Lit}_A(\tau)$ , we have that  $\mathfrak{A}_\pi \models L$  if, and only if,  $\pi(L) \neq 0$ .

A  $K$ -interpretation  $\pi: \text{Lit}_A(\tau) \rightarrow K$  extends in a straightforward way to a valuation  $\pi[\![\varphi(\bar{a})]\!]$  of any instantiation of a formula  $\varphi(\bar{x}) \in \text{FO}(\tau)$ , assumed to be written in negation normal form, by a tuple  $\bar{a} \subseteq A$ . The semiring semantics  $\pi[\![\varphi(\bar{a})]\!]$  is defined by induction. We first extend  $\pi$  by mapping equalities and inequalities to their truth values, by setting  $\pi[\![a = a]\!] := 1$  and  $\pi[\![a = b]\!] := 0$  for  $a \neq b$  (and analogously for inequalities). Further, disjunctions and existential quantifiers are interpreted as sums, and conjunctions and universal quantifiers as products:

<sup>1</sup> In the following, *semiring* always refers to a commutative semiring.

$$\begin{aligned} \pi[\psi(\bar{a}) \vee \vartheta(\bar{a})] &:= \pi[\psi(\bar{a})] + \pi[\vartheta(\bar{a})] & \pi[\psi(\bar{a}) \wedge \vartheta(\bar{a})] &:= \pi[\psi(\bar{a})] \cdot \pi[\vartheta(\bar{a})] \\ \pi[\exists x \vartheta(\bar{a}, x)] &:= \sum_{a \in A} \pi[\vartheta(\bar{a}, a)] & \pi[\forall x \vartheta(\bar{a}, x)] &:= \prod_{a \in A} \pi[\vartheta(\bar{a}, a)]. \end{aligned}$$

Since negation does not correspond to a semiring operation, we insist on writing all formulae in negation normal form. This is a standard approach in semiring semantics (cf. [16]). Equivalence of formulae now takes into account the semiring values and is thus more fine-grained than Boolean equivalence.

► **Definition 1** ( $\equiv_K$ ). *Two formulae  $\psi(\bar{x})$ ,  $\varphi(\bar{x})$  are  $K$ -equivalent (denoted  $\psi \equiv_K \varphi$ ) if  $\pi[\psi(\bar{a})] = \pi[\varphi(\bar{a})]$  for every model-defining  $K$ -interpretation  $\pi$  (over finite universe) and every tuple  $\bar{a}$ . For a class  $\mathcal{S}$  of semirings, we write  $\psi \equiv_{\mathcal{S}} \varphi$  if  $\psi \equiv_K \varphi$  holds for all  $K \in \mathcal{S}$ .*

Towards locality properties, we define distances between two elements  $a, b$  in a  $K$ -interpretation  $\pi$  based on the induced structure  $\mathfrak{A}_\pi$ .

► **Definition 2** (Gaifman graph). *The Gaifman graph  $G(\pi)$  of a model-defining  $K$ -interpretation  $\pi: \text{Lit}_A(\tau) \rightarrow K$  is defined as the Gaifman graph  $G(\mathfrak{A}_\pi)$  of the induced  $\tau$ -structure. That is, two elements  $a \neq b$  of  $A$  are adjacent in  $G(\mathfrak{A}_\pi)$  if, and only if, there exists a positive literal  $L = Rc_1 \dots c_r \in \text{Lit}_A(\tau)$  such that  $\pi(L) \neq 0$  and  $a, b \in \{c_1, \dots, c_r\}$ .*

We write  $d(a, b) \in \mathbb{N}$  for the distance of  $a$  and  $b$  in  $G(\pi)$ . We further define the  $r$ -neighbourhood of an element  $a$  in  $\pi$  as  $B_r^\pi(a) := \{b \in A : d(a, b) \leq r\}$ . For a tuple  $\bar{a} \in A^k$  we put  $B_r^\pi(\bar{a}) := \bigcup_{i \leq k} B_r^\pi(a_i)$ .

Locality properties are really meaningful only for semiring interpretations  $\pi: \text{Lit}_A(\tau) \rightarrow K$  that *track only positive information*, which means that  $\pi(\neg L) \in \{0, 1\}$  for each negative literal  $\neg L$ . Indeed, if also negative literals carry non-trivial information, then either these must be taken into account in the definition of what “local” means, which will trivialise the Gaifman graph (making it a clique) so locality would become meaningless, or otherwise local information no longer suffices to determine values of even very simple sentences involving negative literals, such as  $\exists x \exists y \neg Rxy$ . We therefore consider here only  $K$ -interpretations over finite universes which are model-defining and track only positive information.

### 3 Hanf Locality

The first formalisation of locality that we consider is Hanf locality. We present generalisations of both the Hanf locality rank and of Hanf’s locality theorem, where the latter is conditional on algebraic properties of the semirings. One point that requires care in the adaptation of the classical proofs (cf. [9, 24]) is the combination of partial isomorphisms on disjoint and non-adjacent neighbourhoods. In the setting of semiring semantics, this depends on the assumption that  $K$ -interpretations only track positive information.

► **Lemma 3.** *Let  $\pi_A$  and  $\pi_B$  be model-defining  $K$ -interpretations that track only positive information. Let  $\sigma: B_r^{\pi_A}(\bar{a}) \rightarrow B_r^{\pi_B}(\bar{b})$  and  $\sigma': B_r^{\pi_A}(\bar{a}') \rightarrow B_r^{\pi_B}(\bar{b}')$  be two partial isomorphisms between disjoint  $r$ -neighbourhoods in  $\pi_A$  and  $\pi_B$ . If  $d(\bar{a}, \bar{a}') > 2r + 1$  and  $d(\bar{b}, \bar{b}') > 2r + 1$ , then  $(\sigma \cup \sigma'): B_r^{\pi_A}(\bar{a}, \bar{a}') \rightarrow B_r^{\pi_B}(\bar{b}, \bar{b}')$  is also a partial isomorphism.*

In classical Boolean semantics, a formula  $\psi(\bar{x})$  is *Hanf-local with Hanf locality rank  $r$* , if for any two tuples  $\bar{a}$  in  $\mathfrak{A}$  and  $\bar{b}$  in  $\mathfrak{B}$  we have the equivalence that  $\mathfrak{A} \models \psi(\bar{a}) \Leftrightarrow \mathfrak{B} \models \psi(\bar{b})$  whenever there is a bijection  $f: A \rightarrow B$  such that the  $r$ -neighbourhoods  $B_r^{\mathfrak{A}}(\bar{a}, c)$  and

$B_r^{\mathfrak{A}}(\bar{b}, f(c))$  are isomorphic for all  $c \in A$ . It is known that every first-order formula is Hanf-local, with locality rank depending only on the quantifier rank. The proof of this fact [24] relies on an inductive argument which, from given bijections between  $(3r+1)$ -neighbourhoods of  $k$ -tuples, builds bijections between  $r$ -neighbourhoods of  $(k+1)$ -tuples. Based on Lemma 3, the inductive argument can be adapted to semiring semantics to get the following result, which does not assume any specific properties of the underlying semiring (for details, see [6]).

► **Proposition 4** (Hanf locality in semiring semantics). *Let  $K$  be an arbitrary semiring. For every first-order formula  $\varphi(\bar{x})$ , there exists  $r \in \mathbb{N}$ , depending only on the quantifier rank of  $\varphi$ , such that for all model-defining  $K$ -interpretations  $\pi_A, \pi_B$  that track only positive information, and all tuples  $\bar{a}, \bar{b}$  we have that  $\pi_A[\varphi(\bar{a})] = \pi_B[\varphi(\bar{b})]$  whenever there is a bijection  $f: A \rightarrow B$  such that  $B_r^{\pi_A}(\bar{a}, c) \cong B_r^{\pi_B}(\bar{b}, f(c))$  for all  $c \in A$ .*

A much more fundamental result is *Hanf's locality theorem* which provides a sufficient combinatorial criterion for the  $m$ -equivalence of two structures, i.e. for their indistinguishability by sentences of quantifier rank up to  $m$ . We follow the classical proof in [9], which proceeds by showing that Hanf's criterion admits the construction of a back-and-forth system  $(I_j)_{j \leq m}$  which, by the Ehrenfeucht-Fraïssé theorem, implies the  $m$ -equivalence of the two structures. It turns out that this method carries over to  $K$ -interpretations precisely in the case that the semiring  $K$  is fully idempotent. We further show that for semirings that are not fully idempotent, there actually are counterexamples to Hanf's locality theorem.

To define back-and-forth systems between  $K$ -interpretations, first notice that the notion of partial isomorphisms generalises in an obvious way to  $K$ -interpretations (cf. [6]).

► **Definition 5** (Back-and-forth system). *Let  $\pi_A$  and  $\pi_B$  be two  $K$ -interpretations and let  $k \geq 0$ . A  $m$ -back-and-forth system for  $\pi_A$  and  $\pi_B$  is a sequence  $(I_j)_{j \leq m}$  of finite sets of partial isomorphisms between  $\pi_A$  and  $\pi_B$  such that*

- $\emptyset \in I_m$ , and
- for all  $j < m$ , the set  $I_{j+1}$  has back-and-forth extensions in  $I_j$ , i.e., whenever  $\bar{a} \mapsto \bar{b} \in I_{j+1}$  then for every  $c \in A$  there exists  $d \in B$ , and vice versa, such that  $(\bar{a}c) \mapsto (\bar{b}d)$  is in  $I_j$ .

We write  $(I_j)_{j \leq m}: \pi_A \cong_m \pi_B$  if  $(I_j)_{j \leq m}$  is a  $m$ -back-and-forth system for  $\pi_A$  and  $\pi_B$ .

Back-and-forth systems can be seen as algebraic descriptions of winning strategies in Ehrenfeucht-Fraïssé games, and in classical semantics, an  $m$ -back-and-forth system between two structures exists if, and only if, the structures are  $m$ -equivalent. However, in semiring semantics this equivalence may, in general, fail in both directions [7]. A detailed investigation of the relationship between elementary equivalence, Ehrenfeucht-Fraïssé games, and back-and-forth-systems in semiring semantics is outside the scope of this paper, and will be presented in forthcoming work. For the purpose of studying Hanf locality, we shall need just the fact that in the specific case of fully idempotent semirings,  $m$ -back-and-forth systems do indeed provide a sufficient criterion for  $m$ -equivalence.

► **Proposition 6.** *Let  $\pi_A$  and  $\pi_B$  be  $K$ -interpretations into a fully idempotent semiring  $K$ . If there is an  $m$ -back-and-forth system  $(I_j)_{j \leq m}$  for  $\pi_A$  and  $\pi_B$ , then  $\pi_A \equiv_m \pi_B$ .*

**Proof.** We show by induction that for every first-order formula  $\psi(\bar{x})$  of quantifier rank  $j \leq m$  and every partial isomorphism  $\bar{a} \mapsto \bar{b} \in I_j$  we have that  $\pi_A[\psi(\bar{a})] = \pi_B[\psi(\bar{b})]$ . For  $j = 0$  this is trivial. For the inductive case it suffices to consider formulae  $\psi(\bar{x}) = \exists y \varphi(\bar{x}, y)$  and  $\psi(\bar{x}) = \forall y \varphi(\bar{x}, y)$ , and a map  $\bar{a} \mapsto \bar{b} \in I_{j+1}$ . We have that

$$\begin{aligned} \pi_A[\exists y \varphi(\bar{a}, y)] &= \sum_{c \in A} \pi_A[\varphi(\bar{a}, c)] & \text{and} & & \pi_B[\exists y \psi(\bar{b}, y)] &= \sum_{d \in B} \pi_B[\varphi(\bar{b}, d)], \\ \pi_A[\forall y \varphi(\bar{a}, y)] &= \prod_{c \in A} \pi_A[\varphi(\bar{a}, c)] & \text{and} & & \pi_B[\forall y \psi(\bar{b}, y)] &= \prod_{d \in B} \pi_B[\varphi(\bar{b}, d)]. \end{aligned}$$

Since the semiring is fully idempotent, the valuations  $\pi_A[\exists y \varphi(\bar{a}, y)]$  and  $\pi_A[\forall y \varphi(\bar{a}, y)]$  only depend on the *set* of all values  $\pi_A[\varphi(\bar{a}, c)]$  for  $c \in A$ , and not on their multiplicities. It thus suffices to prove that the sets of values are identical for  $(\pi_A, \bar{a})$  and  $(\pi_B, \bar{b})$ , i.e.

$$\{\pi_A[\varphi(\bar{a}, c)] : c \in A\} = \{\pi_B[\varphi(\bar{b}, d)] : d \in B\}.$$

But this follows immediately from the fact that  $\bar{a} \mapsto \bar{b}$  has back and forth extensions in  $I_j$ , and from the induction hypothesis: for each  $c \in A$  there exists some  $d \in B$ , and vice versa, such that the map  $(\bar{a}, c) \mapsto (\bar{b}, d)$  is in  $I_j$ , and therefore  $\pi_A[\varphi(\bar{a}, c)] = \pi_B[\varphi(\bar{b}, d)]$ .  $\blacktriangleleft$

To formulate Hanf's criterion for  $K$ -interpretations  $\pi_A, \pi_B$ , we write  $\pi_A \rightleftharpoons_{r,t} \pi_B$ , for  $r, t \in \mathbb{N}$ , if for every isomorphism type  $\iota$  of  $r$ -neighbourhoods, either  $\pi_A$  and  $\pi_B$  have the same number of realisations of  $\iota$ , or both have at least  $t$  realisations.

► **Theorem 7** (Hanf's theorem for fully idempotent semirings). *Let  $K$  be a fully idempotent semiring. For all  $m, \ell \in \mathbb{N}$  there exist  $r = r(m) \in \mathbb{N}$  and  $t = t(m, \ell) \in \mathbb{N}$  such that for all model-defining  $K$ -interpretations  $\pi_A$  and  $\pi_B$  that track only positive information and whose Gaifman graphs have maximal degree  $\leq \ell$ , we have that  $\pi_A \equiv_m \pi_B$  whenever  $\pi_A \rightleftharpoons_{r,t} \pi_B$ .*

**Proof.** Given  $m, \ell \in \mathbb{N}$ , let  $r_0 = 0$ , inductively define  $r_{i+1} = 3r_i + 1$ , and set  $r = r_{m-1}$ . Further, let  $t = m \cdot e + 1$ , where  $e := 1 + \ell + \ell^2 + \dots + \ell^r$  is the maximal number of elements in an  $r$ -neighbourhood of a point, in  $K$ -interpretations with Gaifman graphs with maximal degree  $\ell$ . Assume that  $\pi_A$  and  $\pi_B$  are  $K$ -interpretations with that property, such that  $\pi_A \rightleftharpoons_{r,t} \pi_B$ .

We construct an  $m$ -back-and-forth system  $(I_j)_{j \leq m}$  for  $(\pi_A, \pi_B)$  by setting

$$I_j := \{\bar{a} \mapsto \bar{b} : |\bar{a}| = |\bar{b}| = m - j \text{ and } B_{r_j}^{\pi_A}(\bar{a}) \cong B_{r_j}^{\pi_B}(\bar{b})\}.$$

We have  $I_m = \{\emptyset\}$ , and since  $\pi_A \rightleftharpoons_{r,t} \pi_B$ , we have for every  $a \in A$  some  $b \in B$ , and vice versa, such that  $B_r^{\pi_A}(a) \cong B_r^{\pi_B}(b)$ , so  $I_m$  has back-and-forth extensions in  $I_{m-1}$ . Consider now a partial isomorphism  $\bar{a} \mapsto \bar{b}$  in  $I_{j+1}$ . There is an isomorphism  $\rho: B_{3r_j+1}^{\pi_A}(\bar{a}) \cong B_{3r_j+1}^{\pi_B}(\bar{b})$ . By symmetry, it suffices to prove the forth-property: for every  $a \in A$  we must find some  $b \in B$  such that  $\bar{a}a \mapsto \bar{b}b \in I_j$  which means that  $B_{r_j}^{\pi_A}(\bar{a}a) \cong B_{r_j}^{\pi_B}(\bar{b}b)$ .

*Case 1 (a close to  $\bar{a}$ ).* If  $a \in B_{2r_j+1}^{\pi_A}(\bar{a})$ , then we choose  $b = \rho(a) \in B_{2r_j+1}^{\pi_B}(\bar{b})$ . This is a valid choice since  $B_{r_j}^{\pi_A}(\bar{a}a) \subseteq B_{3r_j+1}^{\pi_A}(\bar{a})$  so  $\rho$  also provides an isomorphism between  $B_{r_j}^{\pi_A}(\bar{a}a)$  and  $B_{r_j}^{\pi_B}(\bar{b}b)$ .

*Case 2 (a far from  $\bar{a}$ ).* If  $a \notin B_{2r_j+1}^{\pi_A}(\bar{a})$ , then  $B_{r_j}^{\pi_A}(a) \cap B_{r_j}^{\pi_A}(\bar{a}) = \emptyset$ . Hence, it suffices to find  $b \in B$  such that  $B_{r_j}^{\pi_B}(b)$  has the same isomorphism type as  $B_{r_j}^{\pi_A}(a)$  (call this  $\iota$ ) with the property that  $b$  has distance at least  $2r_j + 2$  to  $\bar{b}$ . Since  $\pi_A$  and  $\pi_B$  only track positive information the isomorphisms can be combined by Lemma 3 to show that  $B_{r_j}^{\pi_A}(\bar{a}a) \cong B_{r_j}^{\pi_B}(\bar{b}b)$ .

Assume that no such  $b$  exists. Let  $s$  be the number of elements realising  $\iota$  in  $\pi_B$ . Since all of them have distance at most  $2r_j + 1$  from  $\bar{b}$  and there are at most  $t$  elements in  $r$ -neighbourhoods around  $\bar{b}$ , we have that  $s \leq t$ . On the other side there are at least  $s + 1$  elements realising  $\iota$  in  $\pi_A$ , namely  $s$  elements in  $B_{2r_j+1}^{\pi_A}(\bar{a})$  (due to  $\rho$ ) and  $a$ . But this

contradicts the fact that  $\iota$  either has the same number of realisations in  $\pi_A$  and  $\pi_B$ , or more than  $t$  realisations in both interpretations. Hence such an element  $b$  exists, and we have proved that  $(I_j)_{j \leq m}$  is indeed a  $m$ -back-and-forth system for  $(\pi_A, \pi_B)$ .

By Proposition 6 this implies that  $\pi_A \equiv_m \pi_B$ .  $\blacktriangleleft$

On the other side, we observe that Hanf's locality theorem in general *fails* for semirings with non-idempotent operations.

► **Example 8 (Counterexample Hanf).** Consider the natural semiring  $(\mathbb{N}, +, \cdot, 0, 1)$  and  $\psi = \exists x Ux$  over signature  $\tau = \{U\}$ . For each  $n$ , we define a model-defining  $K$ -interpretation  $\pi_n$  with universe  $\{a_1, \dots, a_n\}$  by setting  $\pi(Ua_i) = 1$  for all  $i$ . Then  $\pi_n \llbracket \psi \rrbracket = \sum_i \pi(Ua_i) = n$ .

As we only have unary predicates, all neighbourhoods are trivial. That is, they consist of just one element and all of them have the same isomorphism type. Thus,  $\pi_n$  realises this single isomorphism type precisely  $n$  times, which means that  $\pi_n \rightleftharpoons_{r,t} \pi_t$  for all  $r, t$  with  $n \geq t$ . But  $\pi_n \llbracket \psi \rrbracket \neq \pi_t \llbracket \psi \rrbracket$  for  $n \neq t$ , so Hanf's theorem fails for the natural semiring.

This example readily generalises to all semirings containing an element  $s \in K$  for which there are arbitrarily large numbers  $n, m \in \mathbb{N}$  with  $m \cdot s \neq n \cdot s$  or  $s^m \neq s^n$  ( $m \cdot s$  and  $s^m$  refer to the  $m$ -fold addition and multiplication of  $s$ , respectively). Indeed, we can map all atoms  $Ua_i$  to  $s$  and observe that Hanf's theorem fails for either  $\psi = \exists x Ux$  or  $\psi = \forall x Ux$ .  $\blacktriangleleft$

## 4 Gaifman Normal Forms in Semirings Semantics

We briefly recall the classical notion of Gaifman normal forms (cf. [11, 9]), which capture locality in a syntactic way. Gaifman normal forms are Boolean combinations of *local formulae*  $\varphi^{(r)}(x)$  and *basic local sentences*. A local formula  $\varphi^{(r)}(x)$  is a formula in which all quantifiers are *relativised* to the  $r$ -neighbourhood of  $x$ , for instance  $\exists y \vartheta(x, y)$  is relativised to  $\exists y (d(x, y) \leq r \wedge \vartheta(x, y))$ . Here,  $d(x, y) \leq r$  asserts that  $x$  and  $y$  have distance  $\leq r$  in the Gaifman graph, which can easily be expressed in first-order logic (in Boolean semantics). A basic local sentence asserts that there exist *scattered* elements, i.e., elements with distinct  $r$ -neighbourhoods, which all satisfy the same  $r$ -local formula:  $\exists x_1 \dots \exists x_m (\bigwedge_{i \neq j} d(x_i, x_j) > 2r \wedge \bigwedge_i \varphi^{(r)}(x_i))$ . By Gaifman's theorem, every formula has an equivalent Gaifman normal form, which intuitively means that it only makes statements about distinct local neighbourhoods.

Moving to semiring semantics, we keep the notion of Gaifman normal forms close to the original one, with two exceptions. First, we only consider formulae in negation normal form. This means that we restrict to *positive* Boolean combinations and, in turn, permit the duals of basic local sentences (i.e., the negations of basic local sentences, in negation normal form). Second and most importantly, we lose the ability to express relativised quantifiers<sup>2</sup> in our logic. Instead, we extend first-order logic by adding relativised quantifiers (*ball quantifiers*) of the form  $Qy \in B_r^\tau(x)$  for  $Q \in \{\exists, \forall\}$  with the following semantics: given a formula  $\varphi(x, y)$ , a  $K$ -interpretation  $\pi: \text{Lit}_A(\tau) \rightarrow K$ , and an element  $a$ , we define

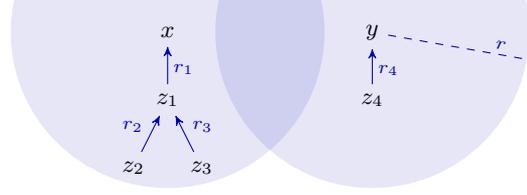
$$\pi \llbracket \exists y \in B_r^\tau(a) \varphi(a, y) \rrbracket := \sum_{b \in B_r^\pi(a)} \pi \llbracket \varphi(a, b) \rrbracket, \quad \pi \llbracket \forall y \in B_r^\tau(a) \varphi(a, y) \rrbracket := \prod_{b \in B_r^\pi(a)} \pi \llbracket \varphi(a, b) \rrbracket.$$

We drop  $\tau$  and write  $\exists y \in B_r(a)$  or  $\forall y \in B_r(a)$  if the signature is clear from the context.

<sup>2</sup> We could use the same formula for  $d(x, y) \leq r$  as in the Boolean case. However, this formula would not just evaluate to 0 or 1, but would include the values of all edges around  $x$ , so each relativised quantifier would have the unintended side-effect of multiplying with the edge values in the neighbourhood. One can show that this side-effect would make Gaifman normal forms impossible (see [6] for details).

$$\varphi^{(r)}(x, y) =$$

$$\forall z_1 \in B_{r_1}(x) (\exists z_2 \in B_{r_2}(z_1) \forall z_3 \in B_{r_3}(z_1) \neg E z_2 z_3) \\ \vee \forall z_4 \in B_{r_4}(y) E x z_4$$



■ **Figure 1** Example of a local formula and the corresponding quantification dag  $D(\varphi)$ , with circles indicating  $B_r(xy)$ . In this example,  $\varphi^{(r)}(x, y)$  is  $r$ -local for all  $r \geq \max(r_1 + r_2, r_1 + r_3, r_4)$ .

This alone is not as expressive as the Boolean notion. For instance, consider  $\varphi^{(r)}(x) = \exists y (d(x, y) \leq \frac{r}{2} \wedge \exists z (d(x, z) \leq r \wedge d(y, z) \leq \frac{r}{2} \wedge \dots))$  which quantifies  $z$  local around  $y$ . Using ball quantifiers, we want to write this as  $\varphi^{(r)}(x) = \exists y \in B_{\frac{r}{2}}(x) (\exists z \in B_{\frac{r}{2}}(y) \dots)$ , so we also permit ball quantifiers around previously quantified variables (here  $y$ ), as long as they stay within the  $r$ -neighbourhood of  $x$  (here:  $\frac{r}{2} + \frac{r}{2} \leq r$ ).

To formalise this condition, we consider the *quantification dag*  $D(\varphi)$  of a formula  $\varphi(\bar{x})$  which contains nodes for all variables in  $\varphi$  and where for every quantifier  $Qz \in B_{r'}(y)$  in  $\varphi$ , we add an edge  $z \rightarrow y$  with distance label  $r'$  (see Figure 1). If the summed distance of any path ending in a free variable  $x \in \bar{x}$  is at most  $r$ , then  $\varphi$  is  $r$ -local.

► **Definition 9** (Local formula). *An  $r$ -local  $\tau$ -formula around  $\bar{x}$ , denoted  $\varphi^{(r)}(\bar{x})$ , is built from  $\tau$ -literals by means of  $\wedge$ ,  $\vee$  and ball quantifiers  $Qz \in B_{r'}(y)$  such that in the associated quantification dag  $D(\varphi)$ , all paths ending in a free variable  $x \in \bar{x}$  have total length at most  $r$ .*

We emphasise that in the Boolean case, Definition 9 is equivalent to the standard notion, so we do not add expressive power. For convenience, we allow quantification  $Qz \in B_{r'}(\bar{y}) \varphi(\bar{y}, z)$  around a tuple  $\bar{y}$ , which can easily be simulated by regular ball quantifiers.

For basic local sentences, we further need to quantify over scattered tuples. To this end, we also add *scattered quantifiers*  $\exists^{r\text{-sc}}(\bar{y})$  and  $\forall^{r\text{-sc}}(\bar{y})$  with the following semantics:

$$\pi[\exists^{r\text{-sc}}(\bar{y}) \varphi(\bar{y})] = \sum_{\substack{\bar{a} \subseteq A \\ d(a_i, a_j) > 2r \text{ for } i \neq j}} \pi[\varphi(\bar{a})], \quad \pi[\forall^{r\text{-sc}}(\bar{y}) \varphi(\bar{y})] = \prod_{\substack{\bar{a} \subseteq A \\ d(a_i, a_j) > 2r \text{ for } i \neq j}} \pi[\varphi(\bar{a})].$$

We remark that in idempotent semirings, which will be the main focus of our positive results, the addition of ball quantifiers makes it possible to express  $d(x, y) \leq r$  by a formula that only assumes values 0 or 1, such as  $\exists x' \in B_{\frac{r}{2}}(x) \exists y' \in B_{\frac{r}{2}}(y) (x' = y')$ , which is  $\frac{r}{2}$ -local around  $xy$ , or alternatively  $\exists x' \in B_r(x) (x' = y)$ , which is  $r$ -local only around  $x$ . Analogously for  $d(x, y) > r$ , so we permit the use of distance formulae to simplify notation whenever we work in idempotent semirings. Scattered quantifiers can then easily be expressed as  $\exists^{r\text{-sc}}(y_1, \dots, y_m) \vartheta(\bar{y}) := \exists y_1 \dots \exists y_m (\bigwedge_{i < j} d(y_i, y_j) > 2r \wedge \vartheta(\bar{y}))$  and  $\forall^{r\text{-sc}}(y_1, \dots, y_m) \vartheta(\bar{y}) := \forall y_1 \dots \forall y_m (\bigvee_{i < j} d(y_i, y_j) \leq 2r \vee \vartheta(\bar{y}))$ .

► **Definition 10** (Local sentence). *A basic local sentence is a sentence of the form*

$$\exists^{r\text{-sc}}(y_1, \dots, y_m) \bigwedge_{i \leq m} \varphi^{(r)}(y_i) \quad \text{or} \quad \forall^{r\text{-sc}}(y_1, \dots, y_m) \bigvee_{i \leq m} \varphi^{(r)}(y_i).$$

A local sentence is a positive Boolean combination of basic local sentences.

Based on these notions we can now formulate precisely the questions about Gaifman normal forms in semiring semantics:

- (1) For which semirings  $K$  does every first-order *sentence* have a  $K$ -equivalent local sentence?
- (2) For which semirings  $K$  is it the case that every first-order *formula* is  $K$ -equivalent to a positive Boolean combination of local formulae and basic local sentences?

## 5 Counterexamples Against Gaifman Normal Forms

This section presents two examples for which a Gaifman normal form does not exist. Both use the vocabulary  $\tau = \{U\}$  with only unary predicates, so that the Gaifman graph  $G(\pi)$  of any  $K$ -interpretation  $\pi: \text{Lit}_A(\tau) \rightarrow K$  is trivial and the  $r$ -neighbourhood of a point, for any  $r$ , consists only of the point itself. Thus, local formulae  $\varphi^{(r)}(x)$  around  $x$  can always be written as positive Boolean combinations of literals  $Ux$ ,  $\neg Ux$  and equalities  $x = x$ ,  $x \neq x$ . Scattered tuples are simply distinct tuples, so we write  $\exists^{\text{distinct}}(\bar{x})$  instead of  $\exists^{r\text{-sc}}(\bar{x})$ .

### 5.1 A Formula Without a Gaifman Normal Form

Consider the formula  $\psi(x) := \exists y(Uy \wedge y \neq x)$  which, in classical Boolean semantics, has the Gaifman normal form  $\varphi(x) := \exists^{\text{distinct}}(y, z)(Uy \wedge Uz) \vee (\neg Ux \wedge \exists y Uy)$ . However, in semiring semantics it is in general not the case that  $\psi(x) \equiv_K \varphi(x)$ . Here we consider the specific case of a universe with two elements  $A = \{a, b\}$  and  $K$ -interpretations  $\pi_{st}$  with  $\pi_{st}(Ua) = s$  and  $\pi_{st}(Ub) = t$ , where  $s, t \in K \setminus \{0\}$  and  $s \neq t$ . Then  $\pi_{st}[\psi(a)] = t$  but  $\pi_{st}[\varphi(a)] = st + ts$ . So, unless  $K$  is the Boolean semiring, we find elements  $s, t$  where  $\pi_{st}[\psi(a)] \neq \pi_{st}[\varphi(a)]$ .

Of course, it might still be the case that there is a different Gaifman normal form of  $\psi(x)$  for semiring interpretations in a specific semiring  $K$ . We prove that this is not the case.

► **Proposition 11.** *In any naturally ordered semiring with at least three elements, the formula  $\psi(x) = \exists y(Uy \wedge y \neq x)$  does not have a Gaifman normal form.*

For the proof, we describe the values that the building blocks of Gaifman normal forms may assume in  $\pi_{st}$ . Recall that a local formula  $\alpha(x)$  is equivalent to a positive Boolean combination of literals  $Ux$ ,  $\neg Ux$ , and equalities. Since  $\pi_{st}(\neg Ux) = 0$  for all  $x \in A$ , we can view the evaluation  $\pi_{st}[\alpha(a)]$  as an expression built from the semiring operations, the value  $\pi_{st}(Ua) = s$  and constants 0, 1. Analogously for  $\pi_{st}[\alpha(b)]$ , but using  $\pi_{st}(Ub) = t$  instead of  $s$ . Hence there is a polynomial  $p_\alpha(X) \in K[X]$  such that  $\pi_{st}[\alpha(a)] = p_\alpha(s)$  and  $\pi_{st}[\alpha(b)] = p_\alpha(t)$ , for all interpretations  $\pi_{st}$ . For the evaluation of a basic local sentence  $\beta = \exists^{\text{distinct}}(y, z)(\alpha(y) \wedge \alpha(z))$ , we then obtain  $\pi_{st}[\beta] = p_\alpha(s)p_\alpha(t) + p_\alpha(t)p_\alpha(s)$ . That is,  $\beta$  can be described by a polynomial  $p_\beta(X, Y) \in K[X, Y]$  such that  $\pi_{st}[\beta] = p_\beta(s, t)$  and  $p_\beta$  is *symmetric* (that is,  $p_\beta(X, Y) = p_\beta(Y, X)$ ). The same holds for universal basic local sentences  $\beta = \forall^{\text{distinct}}(y, z)(\alpha(y) \vee \alpha(z))$ .

Every Gaifman normal form  $\varphi(x)$  can thus be represented by a polynomial  $f_\varphi(X, Y) = \sum_i h_i(X)g_i(X, Y)$ , with symmetric  $g_i$ , such that  $\pi_{st}[\varphi(a)] = f_\varphi(s, t)$  for all  $s, t$ . Proposition 11 then follows from the following algebraic observation (see [6] for a proof).

► **Lemma 12.** *Let  $K$  be a naturally ordered semiring with at least three elements. For any polynomial  $f(X, Y) = \sum_i h_i(X)g_i(X, Y)$  where the  $g_i$  are symmetric polynomials, there exist values  $s, t \in K \setminus \{0\}$  such that  $f(s, t) \neq t$ .*

### 5.2 A Sentence Without a Gaifman Normal Form

While Gaifman normal forms need not exist for formulae, in all relevant semirings beyond the Boolean one, they might still exist for sentences. Indeed, we shall prove a positive result for min-max semirings. However, such a result seems only possible for semirings where both operations are idempotent, similar to Hanf's theorem. For other semirings one can find rather simple counterexamples, as we illustrate for the tropical semiring  $\mathbb{T} = (\mathbb{R}_+^\infty, \min, +, \infty, 0)$ .

► **Proposition 13.** *The sentence  $\psi := \exists z \forall x \exists y(Uy \vee x = z)$  has no Gaifman normal form in the tropical semiring.*

The proof again works by describing the values of basic local sentences, this time in  $\mathbb{T}$ -interpretations of increasingly large size. One can then show that these values are either constant or grow too fast, compared to the value of  $\psi$  (see [6] for details). A similar construction works for the natural semiring  $(\mathbb{N}, +, \cdot, 0, 1)$  and we conjecture that it can be adapted to any infinite semiring with operations that are not idempotent.

## 6 Gaifman's Theorem for Min-Max Semirings

In this section, we prove our main result: a version of Gaifman's theorem for sentences evaluated in min-max semirings (which can be lifted to lattice semirings, see Sect. 7). We write  $\mathcal{M}$  for the class of min-max semirings and refer to  $\equiv_{\mathcal{M}}$  as *minmax-equivalence*.

► **Theorem 14** (Gaifman normal form). *Let  $\tau$  be a finite relational signature. Every  $\text{FO}(\tau)$ -sentence  $\psi$  is minmax-equivalent ( $\equiv_{\mathcal{M}}$ ) to a local sentence.*

Contrary to Hanf's locality theorem, we cannot follow the classical proofs of Gaifman's theorem. For instance, the proof in [9] is based on the Ehrenfeucht-Fraïssé method and makes use of characteristic sentences, which in general do not exist in semiring semantics over min-max semirings (cf. [15]). Gaifman's original proof [11] is a constructive quantifier elimination argument (which is similar to our approach), but makes use of negation to encode case distinctions in the formula, which is not possible in semiring semantics. Another argument why Gaifman's proof does not go through is that it applies to formulae, whereas formulae need not have Gaifman normal forms in our setting (cf. Sect. 5.1).

Instead, we present a novel proof of Gaifman's theorem that applies to the Boolean case as well as to min-max semirings. While our strategy is similar to Gaifman's – a constructive elimination of quantifier alternations – we have to phrase all results in terms of sentences and need to be more careful to derive equivalences that hold in all min-max semirings. These restrictions lead to a slight strengthening of Gaifman's classical result (see Sect. 7).

### 6.1 Toolbox

The proof is rather technical, but is based on a few simple observations. First notice that min-max semirings share many algebraic properties with the Boolean semiring. As a consequence, many classical logical equivalences are also minmax-equivalences, such as distributivity or idempotence. In particular, we can make use of disjunctive normal forms, conjunctive normal forms and prenex normal forms. Moreover, we can exploit the inherent symmetry of min-max semirings to simplify our proofs: arguments for existential sentences can be dualised for universal sentences (see [6] for details). However, we still have to consider quantifier alternations, which pose the main challenge.

Concerning locality, we make two simple but crucial observations. For the first one, consider a local formula  $\varphi^{(r)}(x, y)$  around two variables  $x$  and  $y$ . Such a formula may assert that  $x$  and  $y$  are close to each other, for instance  $\varphi^{(r)}(x, y) = Exy$ . But if  $x$  and  $y$  do not occur together within one literal, then  $\varphi^{(r)}$  intuitively makes independent statements about the neighbourhood of  $x$ , and the neighbourhood of  $y$ , so we can split  $\varphi^{(r)}$  into two separate local formulae. For the general case  $\varphi^{(r)}(\bar{x})$  in several variables, we group  $\bar{x}$  into tuples  $\bar{x}^1, \dots, \bar{x}^n$  with the idea that  $\varphi^{(r)}$  makes independent statements about each group  $\bar{x}^i$ .

► **Lemma 15** (Separation). *Let  $\varphi^{(r)}(\bar{x}^1, \dots, \bar{x}^n)$  be a local formula around  $\bar{x}^1 \dots \bar{x}^n$  and define  $X_i$  as the set of variables connected to some  $x \in \bar{x}^i$  in  $D(\varphi)$ . If each literal of  $\varphi^{(r)}(\bar{x}^1, \dots, \bar{x}^n)$  uses only variables in  $\bar{x}^i \cup X_i$  for a single  $i$ , then  $\varphi^{(r)}(\bar{x}^1, \dots, \bar{x}^n)$  is minmax-equivalent to a positive Boolean combination of  $r$ -local formulae around a single group  $\bar{x}^i$ .*

The second observation is that we can perform a clustering of any tuple  $(a_1, \dots, a_n) \in A^n$  into classes  $I_1, \dots, I_k$  so that elements within one class have “small” distance to each other, whereas different classes are “far apart”. This simple combinatorial observation is a fruitful tool to construct Gaifman normal forms: it becomes easy to quantify elements with a known clustering, and by the following lemma we can then do a disjunction over all clusterings.

► **Definition 16** (Configuration). *Let  $\pi$  be a  $K$ -interpretation with universe  $A$ . Let  $P = \{I_1, \dots, I_k\}$  be a partition of  $\{1, \dots, n\}$  and define representatives  $i_l = \min I_l$  of each class. We say that a tuple  $(a_1, \dots, a_n) \in A^n$  is in configuration  $(P, r)$ , if*

- (a)  $d(a_{i_l}, a_{i_{l'}}) \leq 5^{n-k}r - r$ , for all  $i \in I_l, l \in \{1, \dots, k\}$ , and
- (b)  $d(a_{i_l}, a_{i_{l'}}) > 4 \cdot 5^{n-k}r$ , for all  $l \neq l'$  (representatives are  $(2 \cdot 5^{n-k}r)$ -scattered).

Such a partition always exists: condition (a) remains true if we merge two classes violating (b), so starting from  $P = \{\{1\}, \dots, \{n\}\}$  we can merge classes until (b) holds.

► **Lemma 17** (Clustering). *Let  $\pi$  be a  $K$ -interpretation on  $A$ . For all tuples  $(a_1, \dots, a_n) \in A^n$  and all  $r \geq 1$ , there is a partition  $P$  such that  $(a_1, \dots, a_n)$  is in configuration  $(P, r)$ .*

## 6.2 Proof Outline for Gaifman’s Theorem

The heart of our proof is the elimination of quantifier alternations. Due to space reasons, we refer to the full version [6] for detailed proofs. Here we present an overview of the main steps. Each step proves, building on the previous ones, that sentences of a certain fragment can be translated to minmax-equivalent local sentences. These fragments consist of

- (1) sentences of the form  $\exists^{r\text{-sc}}(x_1, \dots, x_m) \bigwedge_{i \leq m} \varphi_i^{(r)}(x_i)$ ;
- (2) existential sentences  $\exists \bar{x} \varphi^{(r)}(\bar{x})$ ;
- (3) existential-universal sentences  $\exists \bar{y} \forall \bar{x} \varphi^{(r)}(\bar{y}, \bar{x})$ ;
- (4) all first-order sentences (Theorem 14).

We first note that Theorem 14 (step (4)) is a rather simple consequence of (2) and (3). By applying (3) and putting the resulting local sentence in prenex normal form, we can bring  $\exists^* \forall^*$ -sentences into  $\forall^* \exists^*$ -form. We can thus inductively<sup>3</sup> eliminate quantifier alternations by swapping quantifiers, until at most one alternation remains and (2) or (3) apply directly.

For step (1), note that the difference to a basic local sentence is that we permit different local formulae  $\varphi_i^{(r)}$  for each  $x_i$  (such sentences have been called *asymmetric* in [20, 8]). Our proof is an inductive construction of the equivalent local sentence. This step is quite technical, but greatly simplifies the following constructions.

To prove (2), we have to rewrite the  $\exists^*$ -prefix as a scattered quantifier  $\exists^{r\text{-sc}}(\bar{x})$ . This essentially follows from the Clustering and Separation Lemmas: for a given partition  $P = \{I_1, \dots, I_k\}$  we can do a scattered quantification of the representatives  $x_{i_1}, \dots, x_{i_k}$ , and then quantify the elements of each class  $I_l$  locally around its representative  $x_{i_l}$ .

Step (3) is the core of the elimination argument and the most difficult step of the proof. We roughly follow the structure of Gaifman’s proof [11] and, for a sentence  $\exists \bar{y} \forall \bar{x} \varphi^{(r)}(\bar{y}, \bar{x})$ , first split  $\forall \bar{x}$  into those elements close to  $\bar{y}$  (which we can quantify locally within  $\varphi$ ) and those elements far from  $\bar{y}$ , using the Separation Lemma. Eventually, we arrive at a positive Boolean combination of sentences  $\exists \bar{y} (\varphi_{\text{close}}^{(r)}(\bar{y}) \wedge \forall x \notin B_s(\bar{y}) \varphi_{\text{far}}^{(s)}(x))$ . Here, the far elements

<sup>3</sup> An attentive reader may notice that we have to deal with formulae with free variables in the induction, but (2), (3) only apply to sentences. We resolve this issue by temporarily substituting atoms with free variables by fresh relation symbols, without affecting the Gaifman graph (see [6, Abstraction Lemma]).

are covered by the *outside quantifier*  $\forall x \notin B_s(\bar{y})$  with the obvious semantics. As in Gaifman’s proof, the main challenge is the elimination of this outside quantifier. Gaifman approaches this by using negation to encode case distinctions in the Gaifman normal form. Our proof instead consists of a series of surprisingly difficult syntactical transformations that avoid negation, eventually leading to a minmax-equivalent local sentence without outside quantifiers.

## 7 Strengthening Gaifman’s Theorem

In this section, we rephrase our main result in terms of Boolean semantics, which leads to a novel strengthening of Gaifman’s classical theorem. Interestingly, Theorem 14 can be regained from the Boolean result by algebraic techniques, and even lifted to lattice semirings. These insights suggest that the merit of our proof, and the reason why it is more complicated than Gaifman’s original proof, is the construction of a Gaifman normal form without the use of negation. Since our proof applies in particular to the Boolean semiring and hence to standard Boolean semantics (the only difference is that we use ball quantifiers instead of distance formulae, but these are interchangeable), we obtain the following corollary.

► **Corollary 18** (Gaifman normal form without negation). *Let  $\tau$  be a finite relational signature. In Boolean semantics, every  $\text{FO}(\tau)$ -sentence  $\psi$  has an equivalent local sentence  $\psi'$  such that every relation symbol occurring only positively (only negatively) in  $\psi$  also occurs only positively (only negatively) in  $\psi'$ , not counting occurrences within distance formulae.*

We believe that this result may be of independent interest. A similar adaptation of Gaifman’s theorem has been considered in [20], namely that *existential* sentences are equivalent to *positive* Boolean combinations of *existential* basic local sentences. Our proof of step (2) implies a similar result (cf. [6]), as we also construct a positive Boolean combination of existential basic local sentences. However, we permit distance formulae  $d(x, y) > 2r$  within local formulae (which are abbreviations for universal quantifiers), while [20] does not. Moreover, the approximation schemes of [8] are based on a version of Gaifman’s theorem for sentences positive in a single unary relation (i.e., no negations are added in front of this relation). Their proof uses a version of Ehrenfeucht-Fraïssé games, which is quite different from our syntactical approach. Since unary relations do not occur in distance formulae, Corollary 18 subsumes their result. Interestingly, [20, 8] both share our observation that the proof of the respective version of Gaifman’s theorem is surprisingly difficult.

To prove Theorem 14 from Corollary 18, one can show (cf. [6]) that with some preparation, Boolean equivalences  $\equiv_{\mathbb{B}}$  can be lifted to lattice-equivalences  $\equiv_{\mathcal{L}}$  (which subsume  $\equiv_{\mathcal{M}}$ ). This is done by applying *separating homomorphisms* of [15] to turn a falsifying  $K$ -interpretation  $\pi$ , witnessing  $\not\equiv_{\mathcal{L}}$ , into a falsifying Boolean structure  $h \circ \pi$ , witnessing  $\not\equiv_{\mathbb{B}}$ . Such homomorphisms  $h$  exist for all min-max semirings [15] and also for the more general lattice semirings [7, 6]. We obtain the following generalisation of Theorem 14 by lifting the Boolean result.

► **Corollary 19.** *Let  $\tau$  be a finite relational signature. Every  $\text{FO}(\tau)$ -sentence  $\psi$  is lattice-equivalent ( $\equiv_{\mathcal{L}}$ ) to a local sentence.*

We remark that the lifting argument implies that for many sentences (to be precise, those where no relation occurs both positively and negatively, cf. [6]), the Gaifman normal form in min-max and lattice semirings coincides with the one for Boolean semantics in Corollary 18 (but not necessarily with Gaifman’s original construction). A further consequence is that the counterexample for formulae in Sect. 5.1 also applies to Corollary 18.

## 8 Conclusion

Semiring semantics is a refinement of classical Boolean semantics, which provides more detailed information about a logical statement than just its truth or falsity. This leads to a finer distinction between formulae: statements that are equivalent in the Boolean sense may have different valuations in semiring interpretations, depending on the underlying semiring. It is an interesting and non-trivial question, which logical equivalences and, more generally, which model-theoretic methods, can be carried over from classical semantics to semiring semantics, and how this depends on the algebraic properties of the underlying semiring.

Here we have studied this question for locality properties of first-order logic, in particular for Hanf's locality theorem and for Gaifman normal forms. Our setting assumes semiring interpretations which are model-defining and track only positive information, since these are the conditions that provide well-defined and meaningful locality notions. However, from the outset, it has been clear that one cannot expect to transfer all locality properties of first-order logic to semiring semantics in arbitrary commutative semirings. Indeed, semiring semantics evaluates existential and universal quantifiers by sums and products over all elements of the universe, which gives an inherent source of non-locality if these operations are not idempotent.

Most positive locality results thus require that the underlying semirings are fully idempotent. Under this assumption, one can adapt the classical proof of Hanf's locality theorem to the semiring setting, relying on a back-and-forth argument that itself requires fully idempotent semirings. The question whether there exist Gaifman normal forms in semiring semantics turned out to be more subtle. Indeed, for formulae with free variables Gaifman normal forms need not exist once one goes beyond the Boolean semiring. Also for sentences, one can find examples that do not admit Gaifman normal forms in semirings that are not fully idempotent. We have presented such an example for the tropical semiring.

Our main result, however, is a positive one and establishes the existence of Gaifman normal forms over the class of all min-max and lattice semirings. Intuitively, it relies on the property that in min-max semirings, the value of a quantified statement  $\exists x \varphi(x)$  or  $\forall x \varphi(x)$  coincides with a value of  $\varphi(a)$ , for some witness  $a$ . This needs, for instance, not be the case in lattice semirings, and hence the generalisation to lattice semirings uses a different approach based on separating homomorphisms. It is still an open question whether, in analogy to Hanf's theorem, Gaifman normal forms exist over all fully idempotent semirings. The proof of our main result, which is based on quantifier elimination arguments, turned out to be surprisingly difficult; we identified the lack of a classical negation operator as the main reason for its complexity. An interesting consequence of this restriction is a stronger version of Gaifman's classical theorem in Boolean semantics: every sentence has a Gaifman normal form which, informally speaking, does not add negations.

For applications such as provenance analysis, min-max semirings are relevant, for instance, for studying access levels and security issues. A much larger interesting class of semirings with wider applications are the absorptive ones, including the tropical semiring, in which addition is idempotent, but multiplication in general is not. We have seen that Gaifman normal forms for such semirings need not exist for all sentences. The question arises whether one can establish weaker locality properties for absorptive semirings, applicable perhaps to just a relevant fragment of first-order logic.

---

## References

- 1 M. Arenas, P. Barceló, and L. Libkin. Game-based notions of locality over finite models. *Ann. Pure Appl. Log.*, 152(1-3):3–30, 2008. doi:10.1016/j.apal.2007.11.012.
- 2 M. Benedikt, T. Griffin, and L. Libkin. Verifiable properties of database transactions. *Inf. Comput.*, 147(1):57–88, 1998. doi:10.1006/inco.1998.2731.

- 3 S. van Bergerem. Learning concepts definable in first-order logic with counting. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785811.
- 4 S. van Bergerem, M. Grohe, and M. Ritzert. On the parameterized complexity of learning first-order logic. In *PODS '22: International Conference on Management of Data*, pages 337–346. ACM, 2022. doi:10.1145/3517804.3524151.
- 5 S. van Bergerem and N. Schweikardt. Learning concepts described by weight aggregation logic. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021*, pages 10:1–10:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.10.
- 6 C. Bizière, E. Grädel, and M. Naaf. Locality theorems in semiring semantics, 2023. Full version of this paper. arXiv:2303.12627.
- 7 S. Brinke. *Ehrenfeucht-Fraïssé Games for Semiring Semantics*. M. Sc. thesis, RWTH Aachen University, 2023. URL: <https://logic.rwth-aachen.de/pub/brinke/masterarbeit.pdf>.
- 8 A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Approximation schemes for first-order definable optimisation problems. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 411–420. IEEE, 2006. doi:10.1109/LICS.2006.13.
- 9 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1995. doi:10.1007/3-540-28788-4.
- 10 J. Foster, T. Green, and V. Tannen. Annotated XML: queries and provenance. In *Proceedings of PODS*, pages 271–280, 2008. doi:10.1145/1376916.1376954.
- 11 H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982. doi:10.1016/S0049-237X(08)71879-2.
- 12 B. Glavic. Data provenance. *Foundations and Trends in Databases*, 9(3-4):209–441, 2021. doi:10.1561/19000000068.
- 13 E. Grädel, H. Helal, M. Naaf, and R. Wilke. Zero-one laws and almost sure valuations of first-order logic in semiring semantics. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 41:1–41:12. ACM, 2022. doi:10.1145/3531130.3533358.
- 14 E. Grädel, N. Lücking, and M. Naaf. Semiring provenance for Büchi games: Strategy analysis with absorptive polynomials. In *Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2021)*, volume 346 of *EPTCS*, pages 67–82, 2021. doi:10.4204/EPTCS.346.5.
- 15 E. Grädel and L. Mrkonjić. Elementary equivalence versus isomorphism in semiring semantics. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 133:1–133:20, 2021. doi:10.4230/LIPIcs.ICALP.2021.133.
- 16 E. Grädel and V. Tannen. Semiring provenance for first-order model checking, 2017. arXiv:1712.01980.
- 17 E. Grädel and V. Tannen. Provenance analysis for logic and games. *Moscow Journal of Combinatorics and Number Theory*, 9(3):203–228, 2020. doi:10.2140/moscow.2020.9.203.
- 18 T. Green and V. Tannen. The semiring framework for database provenance. In *Proceedings of PODS*, pages 93–99. ACM, 2017. doi:10.1145/3034786.3056125.
- 19 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
- 20 M. Grohe and S. Wöhrle. An existential locality theorem. *Ann. Pure Appl. Log.*, 129(1-3):131–148, 2004. doi:10.1016/j.apal.2004.01.005.
- 21 L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *J. Symb. Log.*, 64(4):1751–1773, 1999. doi:10.2307/2586810.
- 22 L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. *J. ACM*, 48(4):880–907, 2001. doi:10.1145/502090.502100.
- 23 D. Kuske and N. Schweikardt. Gaifman normal forms for counting extensions of first-order logic. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 133:1–133:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.133.

- 24 L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.
- 25 T. Schwentick and K. Barthelmann. Local normal forms for first-order logic with applications to games and automata. In *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 1998. doi:10.1007/BFb0028580.
- 26 J. Xu, W. Zhang, A. Alawini, and V. Tannen. Provenance analysis for missing answers and integrity repairs. *IEEE Data Eng. Bull.*, 41(1):39–50, 2018. URL: <http://sites.computer.org/debull/A18mar/p39.pdf>.



# A Characterisation of Functions Computable in Polynomial Time and Space over the Reals with Discrete Ordinary Differential Equations

## Simulation of Turing Machines with Analytic Discrete ODEs

Manon Blanc ✉

Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

Olivier Bournez ✉

Institut Polytechnique de Paris, Ecole Polytechnique, LIX, Palaiseau, France

---

### Abstract

We prove that functions over the reals computable in polynomial time can be characterised using discrete ordinary differential equations (ODE), also known as finite differences. We also provide a characterisation of functions computable in polynomial space over the reals. In particular, this covers space complexity, while existing characterisations were only able to cover time complexity, and were restricted to functions over the integers, and we prove that no artificial sign or test function is needed even for time complexity. At a technical level, this is obtained by proving that Turing machines can be simulated with analytic discrete ordinary differential equations. We believe this result opens the way to many applications, as it opens the possibility of programming with ODEs, with an underlying well-understood time and space complexity.

**2012 ACM Subject Classification** Theory of computation → Models of computation; Theory of computation → Computability; Theory of computation → Complexity classes; Mathematics of computing → Ordinary differential equations

**Keywords and phrases** Discrete ordinary differential equations, Finite Differences, Implicit complexity, Recursion scheme, Ordinary differential equations, Models of computation, Analog Computations

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.21

## 1 Introduction

Recursion schemes constitute a major approach to classical computability theory and, to some extent, to complexity theory. The foundational characterisation of **FPTIME**, based on bounded primitive recursion on notations, due to Cobham [10] gave birth to the field of *implicit complexity* at the interplay of logic and theory of programming. Alternative characterizations, based on safe recursion [1] or on ramification ([16, 15]) or for other classes [17] followed: see [8, 9] for monographs.

Initially motivated to help understanding how analogue models of computations compare to classical digital ones, in an orthogonal way, various computability and complexity classes have been recently characterised using Ordinary Differential Equations (ODE). An unexpected side effect of these proofs is the possibility of programming with classical ODEs, over the continuum. It recently led to solving various open problems. This includes the proof of the existence of a universal ODE [6], the proof of the Turing-completeness of chemical reactions [11], or hardness of problems related to dynamical systems [12].

Discrete ODEs, that we consider in this article, are an approach in-between born from the attempt of [4, 5] to explain some of the constructions for continuous ODEs in an easier way. The basic principle is, for a function  $\mathbf{f}(x)$ , to consider its discrete derivative defined as  $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$  (also denoted  $\mathbf{f}'(x)$  in what follows to help analogy with classical continuous counterparts). A consequence of this attempt is the characterisation obtained



© Manon Blanc and Olivier Bournez;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 21; pp. 21:1–21:15

Leibniz International Proceedings in Informatics



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

in [4, 5]. They provided a characterisation of **FPTIME** for functions over the integers that does not require the specification of an explicit bound in the recursion, in contrast to Cobham’s work [10], nor the assignment of a specific role or type to variables, in contrast to safe recursion or ramification [1, 14]. Instead, they only assume involved ODEs to be linear, a very classical natural concept for differential equations.

► **Remark 1.** Unfortunately, even if it was the original motivation, both approaches for characterising complexity classes for continuous and discrete ODEs are currently not directly connected. A key difference is that there is no simple expression (no analogue of the Leibniz rule) for the derivative of the composition of functions in the discrete settings. The Leibniz rule is a very basic tool for establishing results over the continuum, using various stability properties, but that cannot be used easily over discrete settings.

In the context of algebraic classes of functions, the following notation is classical: call *operation* an operator that takes finitely many functions and returns some new function defined from them. Then  $[f_1, f_2, \dots, f_k; op_1, op_2, \dots, op_\ell]$  denotes the smallest set of functions containing  $f_1, f_2, \dots, f_k$  that is closed under the operations  $op_1, op_2, \dots, op_\ell$ . Call *discrete function* a function of type  $f : S_1 \times \dots \times S_d \rightarrow S'_1 \times \dots \times S'_d$ , where each  $S_i, S'_i$  is either  $\mathbb{N}$  or  $\mathbb{Z}$ . Write **FPTIME** for the class of functions computable in polynomial time, and **FPSPACE** for the class of functions computable in polynomial space.

► **Remark 2.** The literature considers two possible definitions for **FPSPACE**, according to whether functions with non-polynomial size values are allowed or not. In our case, we should add “whose outputs remain of polynomial size”, to resolve the ambiguity<sup>1</sup>.

A main result of [4, 5] is the following (LDL stands for linear derivation on length):

► **Theorem 3 ([4]).** *For functions over the reals, we have  $\text{LDL} = \mathbf{FPTIME}$  where  $\text{LDL} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \text{sg}(x) ; \text{composition}, \text{linear length ODE}]$ .*

In particular, writing as usual  $B^A$  for functions from  $A$  to  $B$ , we deduce:

► **Corollary 4 (Functions over the integers).**  $\text{LDL} \cap \mathbb{N}^{\mathbb{N}} = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$ .

That is to say, LDL (and hence **FPTIME** for functions over the integers) is the smallest class of functions that contains the constant functions  $\mathbf{0}$  and  $\mathbf{1}$ , the projections  $\pi_i^k$  of the  $i^e$  coordinate of a vector of size  $k$ , the length function  $\ell(x)$ , mapping an integer to the length of its binary representation, the addition  $x+y$ , the subtraction  $x-y$ , the multiplication  $x \times y$ , the **sign** function  $\text{sg}(x)$  and that is closed under composition (when defined) and linear length-ODE scheme: the linear length-ODE scheme, formally given by Definition 15, corresponds to defining a function from a linear ODEs with respect to derivation along the length of the argument, so of the form  $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{A}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}]$ .

Here, we use the notation  $\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell}$  which corresponds to the derivation of  $\mathbf{f}$  along the length function: given some function  $\mathcal{L} : \mathbb{N}^{p+1} \rightarrow \mathbb{Z}$  and in particular for the case where  $\mathcal{L}(x, \mathbf{y}) = \ell(x)$ ,

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}(x, \mathbf{y})} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \tag{1}$$

is a formal synonym for  $\mathbf{f}(x+1, \mathbf{y}) = \mathbf{f}(x, \mathbf{y}) + (\mathcal{L}(x+1, \mathbf{y}) - \mathcal{L}(x, \mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y})$ .

<sup>1</sup> Otherwise, the class is not closed by composition: this may be considered as a basic requirement when talking about the complexity of functions. The issue is about the usage of not counting the output as part of the total space used. In this model, given  $f$  computable in polynomial space, and  $g$  in logarithmic space,  $f \circ g$  (and  $g \circ f$ ) is computable in polynomial space. But this is not true, if we assume only  $f$  and  $g$  to be computable in polynomial space, since the first might give an output of exponential size.

► **Remark 5.** This concept introduced in [4, 5], is motivated by the fact that the latter expression is similar to the classical formula for continuous ODEs:

$$\frac{\delta f(x, \mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x, \mathbf{y})}{\delta x} \cdot \frac{\delta f(x, \mathbf{y})}{\delta \mathcal{L}(x, \mathbf{y})},$$

and hence is similar to a change of variable. Consequently, a linear length-ODE is basically a linear ODE over a variable  $t$  once the change of variable  $t = \ell(x)$  is done.

However, in the context of (classical) ODEs, considering functions over the reals is more natural than only functions over the integers. Call *real function* a function  $f : S_1 \times \dots \times S_d \rightarrow S'_1 \times \dots \times S'_d$ , where each  $S_i, S'_i$  is either  $\mathbb{R}, \mathbb{N}$  or  $\mathbb{Z}$ . A natural question about the characterisation of **FPTIME** for real functions arises, and not only discrete functions: we consider here computability over the reals in its most classical approach, namely computable analysis [19].

As a first step, the class  $\mathbb{LDDL}^\bullet = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2}; \text{composition, linear length ODE}]$  has been considered in [3, 2] where some characterisation of **P**TIME was obtained, but only for functions from the integers to the reals (i.e. sequences) while it would be more natural to characterise functions from the reals to the reals. More importantly, this was obtained by assuming that some **non-analytic exact function** is among the basic available functions to simulate a Turing machine:  $\overline{\text{cond}}$  valuing 1 for  $x > \frac{3}{4}$  and 0 for  $x < \frac{1}{4}$ .

We prove first this is not needed, and mainly, we extend all previous results to real functions, furthermore covering not only time complexity but also space complexity. Consider

$$\mathbb{LDDL}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE}],$$

where  $\ell : \mathbb{N} \rightarrow \mathbb{N}$  is the length function, mapping some integer to the length of its binary representation,  $\frac{x}{2} : \mathbb{R} \rightarrow \mathbb{R}$  is the function dividing by 2 (similarly for  $\frac{x}{3}$ ) and all other basic functions defined exactly as for  $\mathbb{LDDL}$ , but considered here as functions from the reals to reals.

► **Remark 6.** This class is  $\mathbb{LDDL}$  but without the  $\text{sg}(x)$  function, nor the multiplication function, or  $\mathbb{LDDL}^\bullet$  but without the  $\overline{\text{cond}}$  function, nor the multiplication. This is done by adding the analytic  $\tanh$  functions as a substitute (and adding  $x/3$ ).

► **Remark 7.** We can consider  $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$  but as functions may have different types of outputs, the composition is an issue. We consider, as in [3, 2], that composition may not be defined in some cases: it is a partial operator. For example, given  $f : \mathbb{N} \rightarrow \mathbb{R}$  and  $g : \mathbb{R} \rightarrow \mathbb{R}$ , the composition of  $g$  and  $f$  is defined as expected, but  $f$  cannot be composed with a function such as  $h : \mathbb{N} \rightarrow \mathbb{N}$ .

First, we improve Theorem 3 by stating **FPTIME** over the integers can be characterised algebraically using linear length ODEs and only analytic functions (i.e. no need for sign function). Since  $\mathbb{LDDL}^\circ$  is about functions over the reals, and Theorem 3 is about functions over the integers, we need a way to compare these classes. Given a function  $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$  sending every integer  $\mathbf{n} \in \mathbb{N}^d$  to the vicinity of some integer of  $\mathbb{N}^d$ , say at distance less than  $1/4$ , we write  $\text{DP}(f)$  for its discrete part: this is the function from  $\mathbb{N}^d \rightarrow \mathbb{N}^d$  mapping  $\mathbf{n} \in \mathbb{N}^d$  to the integer rounding of  $\mathbf{f}(n)$ . Given a class  $\mathcal{C}$  of such functions, we write  $\text{DP}(\mathcal{C})$  for the class of the discrete parts of the functions of  $\mathcal{C}$ .

► **Theorem 8.**  $\text{DP}(\mathbb{LDDL}^\circ) = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$ .

Write  $\overline{\mathbb{LDDL}^\circ}$  for the class obtained by adding some effective limit operation similar to the one considered in [3] to get  $\overline{\mathbb{LDDL}^\bullet}$ . We get a characterization of functions over the reals (and not only sequences as in [3]) computable in polynomial time.

► **Theorem 9** (Generic functions over the reals).  $\overline{\text{LDL}}^\circ \cap \mathbb{R}^\mathbb{R} = \mathbf{FPTIME} \cap \mathbb{R}^\mathbb{R}$   
 More generally:  $\overline{\text{LDL}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$

We also prove that, by adding a robust linear ODE scheme (Definition 18), we get a class  $\text{RLD}^\circ$  (this stands for robust linear derivation) with the similar statements but for  $\mathbf{FPSPACE}$ .

► **Theorem 10.**  $\text{DP}(\text{RLD}^\circ) = \mathbf{FPSPACE} \cap \mathbb{N}^\mathbb{N}.$

► **Theorem 11** (Generic functions over the reals).  $\overline{\text{RLD}}^\circ \cap \mathbb{R}^\mathbb{R} = \mathbf{FPSPACE} \cap \mathbb{R}^\mathbb{R}$   
 More generally:  $\overline{\text{RLD}}^\circ \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$

As far as we know, this is the first time a characterisation of  $\mathbf{FPSPACE}$  with discrete ODEs is provided. If we forget the context of discrete ODEs,  $\mathbf{FPSPACE}$  has been characterised in [18] but using a bounded recursion scheme, i.e. requiring some explicit bound in the spirit of Cobham’s statement [10]. We avoid this issue by considering numerically stable schemes, which are very natural in the context of ODEs.

At a technical level, all our results are obtained by proving Turing machines can be simulated with analytic discrete ODEs in a suitable manner. We believe our constructions could be applied to many other situations, where programming with ODEs is needed.

In Section 2, we recall some basic statements about the theory of discrete ODEs. In Section 3, we establish some properties about particular functions required for our proofs. In Section 4 we prove our main technical result: Turing machines can be simulated using functions from  $\text{LDL}^\circ$ . Section 5 is about converting integers and reals (dyadic) to words of a specific form. Section 6 is about applications of our toolbox. We prove in particular all above theorems.

## 2 Some concepts related to discrete ODEs

In this section, we recall some concepts and definitions from discrete ODEs, either well-known or established in [4, 5, 3]. We consider here that  $\mathbf{tanh}$  is  $\tanh$ , the hyperbolic tangent. The papers [4, 5] use similar definitions with the sign function  $\text{sg}$  and [3] with the piecewise affine function  $\text{cond}$ , that values 1 for  $x > \frac{3}{4}$  and 0 for  $x < \frac{1}{4}$ , instead of  $\tanh$ .

► **Definition 12** ([3]). A  $\mathbf{tanh}$ -polynomial expression  $P(x_1, \dots, x_h)$  is an expression built on  $+$ ,  $-$ ,  $\times$  (often denoted  $\cdot$ ) and  $\mathbf{tanh}$  functions over a set of variables  $V = \{x_1, \dots, x_h\}$  and integer constants.

We need to measure the degree, similarly to the classical notion of degree in polynomial expression, but considering all subterms that are within the scope of a  $\mathbf{tanh}$  function contributes to 0 to the degree.

► **Definition 13** ([3]). The degree  $\text{deg}(x, P)$  of a term  $P$  in  $x \in V$  is defined inductively as follows:  $\text{deg}(x, x) = 1$  and for  $x' \in V \cup \mathbb{Z}$  such that  $x' \neq x$ ,  $\text{deg}(x, x') = 0$ ;  $\text{deg}(x, P + Q) = \max\{\text{deg}(x, P), \text{deg}(x, Q)\}$ ;  $\text{deg}(x, P \times Q) = \text{deg}(x, P) + \text{deg}(x, Q)$ ;  $\text{deg}(x, \mathbf{tanh}(P)) = 0$ . A  $\mathbf{tanh}$ -polynomial expression  $P$  is essentially constant in  $x$  if  $\text{deg}(x, P) = 0$ .

A vectorial function (resp. a matrix or a vector) is said to be a  $\mathbf{tanh}$ -polynomial expression if all its coordinates (resp. coefficients) are, and *essentially constant* if all its coefficients are.

► **Definition 14** ([4, 5, 3]). A  $\mathbf{tanh}$ -polynomial expression  $\mathbf{g}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$  is essentially linear in  $\mathbf{f}(x, \mathbf{y})$  if it is of the form:  $\mathbf{A}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}]$  where  $\mathbf{A}$  and  $\mathbf{B}$  are  $\mathbf{tanh}$ -polynomial expressions essentially constant in  $\mathbf{f}(x, \mathbf{y})$ .

For example, the expression  $P(x, y, z) = x \cdot \tanh(x^2 - z) \cdot y + y^3$  is essentially linear in  $x$ , essentially constant in  $z$  and not linear in  $y$ . The expression:  $z + (1 - \tanh(x)) \cdot (1 - \tanh(-x)) \cdot (y - z)$  is essentially constant in  $x$  and linear in  $y$  and  $z$ .

► **Definition 15** (Linear length ODE [4, 5]). *A function  $\mathbf{f}$  is linear length-ODE definable from  $\mathbf{u}$  essentially linear in  $\mathbf{f}(x, \mathbf{y})$ ,  $\mathbf{g}$  and  $\mathbf{h}$ , if it corresponds to the solution of*

$$f(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \ell} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}). \quad (2)$$

A fundamental fact is that the derivation with respect to length provides a way to do some change of variables:

► **Lemma 16** ([4, 5]). *Assume that (2) holds. Then  $\mathbf{f}(x, \mathbf{y})$  is given by  $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\ell(x), \mathbf{y})$  where  $\mathbf{F}$  is the solution of the initial value problem*

$$\mathbf{F}(1, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial t} = \mathbf{u}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}). \quad (3)$$

This means  $\mathbf{f}(x, \mathbf{y})$  depends only on the length of its first argument:  $\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(2^{\ell(x)}, \mathbf{y})$ . Then (3) can be seen as defining a function (with this latter property) by a recurrence of type

$$\mathbf{f}(2^0, \mathbf{y}) = \mathbf{g}(\mathbf{y}), \quad \text{and} \quad \mathbf{f}(2^{t+1}, \mathbf{y}) = \bar{\mathbf{u}}(\mathbf{f}(2^t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t, \mathbf{y}). \quad (4)$$

for some  $\bar{\mathbf{u}}$  is essentially linear in  $\mathbf{f}(2^t, \mathbf{y})$ . As recurrence (3) is basically equivalent to (2):

► **Corollary 17** (Linear length ODE presented with powers of 2). *A function  $\mathbf{f}$  is linear  $\mathcal{L}$ -ODE definable iff the value of  $\mathbf{f}(x, \mathbf{y})$  depends only on the length of its first argument and satisfies (4), for some  $\mathbf{g}$  and  $\mathbf{h}$ , and  $\bar{\mathbf{u}}$ , essentially linear in  $\mathbf{f}(2^t, \mathbf{y})$ .*

We assume it is easier for our reader to deal with recurrences of the form (4) than with ODEs of the form (2). Consequently, this is how we will describe many functions from now on, starting with some basic functions, authorising compositions, and the above schemes. As an example,  $n \mapsto 2^n$  can easily be defined that way (by  $2^0 = 1$ , and  $2^{n+1} = 2 \cdot 2^n = 2^n + 2^n$ ) and we can produce  $n \mapsto 2^{p(n)}$  for any polynomial  $p$ . For example,  $(n_1, \dots, n_k) \rightarrow 2^{n_1 n_2 \dots n_k}$  can be obtained, using  $k$  such schemes in turn, providing the case of the polynomial  $p(n) = n^k$ .

When talking about space complexity, we will also consider the case where the ODE is not derivated with respect to length but with classical derivation. For functions over the reals an important issue is numerical stability.

► **Definition 18** (Robust linear ODE [4, 5]). *A bounded function  $\mathbf{f}$  is robustly linear ODE definable from  $\mathbf{u}$  essentially linear in  $\mathbf{f}(x, \mathbf{y})$ ,  $\mathbf{g}$  and  $\mathbf{h}$  if:*

1. *it corresponds to the solution of*

$$\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}), \quad (5)$$

2. *where the schema (5) is polynomially numerically stable.*

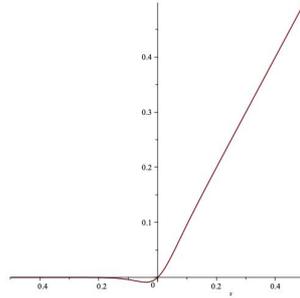
Here, writing  $a =_n b$  for  $\|a - b\| \leq 2^{-n}$  for conciseness, **2.** means formally there exists some polynomial  $p$  such that, for all integer  $n$ , writing  $\epsilon(n) = p(n + \ell(\mathbf{y}))$ , if you consider any solution of  $\tilde{\mathbf{y}} =_{\epsilon(n)} \mathbf{y}$  and  $\tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{h}(x, \tilde{\mathbf{y}})$ , and  $\tilde{\mathbf{f}}(0, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{g}(\mathbf{y})$  and  $\frac{\partial \tilde{\mathbf{f}}(x, \tilde{\mathbf{y}})}{\partial x} =_{\epsilon(n)} \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}})$  then  $\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{\epsilon(n)} \mathbf{f}(x, \mathbf{y})$ .

**3 Some results about various functions**

A key part of our proofs is the construction of very specific functions in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ : we write  $\{x\}$  for the fractional part of the real  $x$ , i.e.  $\{x\} = x - \lfloor x \rfloor$ . We provide some graphical representations of some of them to show that these functions are sometimes highly non-trivial (see for e.g. Figures 3 or 6).

A first observation is that we can uniformly approximate the  $\text{ReLU}(x) = \max(0, x)$  function using a essentially constant function:

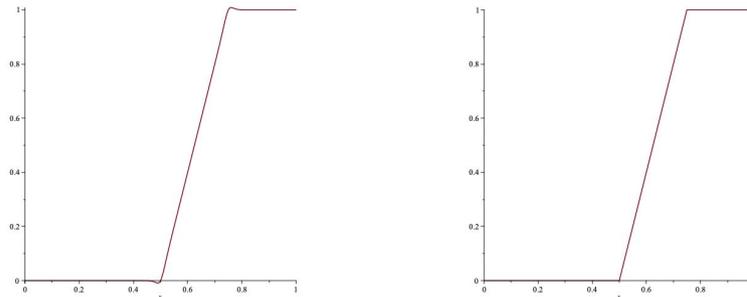
► **Lemma 19.** We denote by  $Y(x, 2^{m+2})$  the function  $Y(x, 2^{m+2}) = \frac{1+\tanh(2^{m+2}x)}{2}$  (illustrated by Figure 1). For all integer  $m$ , for all  $x \in \mathbb{R}$ ,  $|\text{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}$ .



■ **Figure 1** Graphical representation of  $xY(x, 2^{2+2})$  obtained with maple.

We deduce we can uniformly approximate the continuous sigmoid functions (when  $1/(b-a)$  is in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ ) defined as:  $\mathfrak{s}(a, b, x) = 0$  whenever  $w \leq a$ ,  $\frac{x-a}{b-a}$  whenever  $a \leq x \leq b$ , and 1 whenever  $b \leq x$ .

► **Lemma 20 (Uniform approximation of any piecewise continuous sigmoid).** Assume  $\frac{1}{b-a}$  is in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ . Then there is some function  $\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$  (illustrated by Figure 2) such that for all integer  $m$ ,  $|\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) - \mathfrak{s}(a, b, x)| \leq 2^{-m}$ .

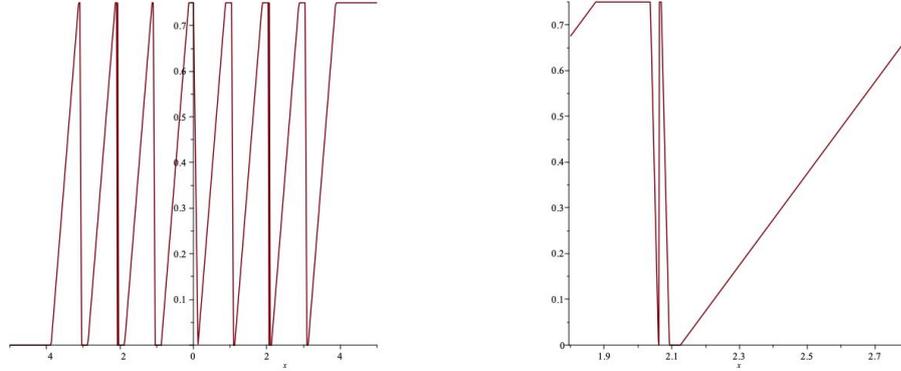


■ **Figure 2** Graphical representation of  $\mathcal{C}\text{-}\mathfrak{s}(2, \frac{1}{2}, \frac{3}{4}, x)$  and  $\mathcal{C}\text{-}\mathfrak{s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$  obtained with maple.

**Proof.** We can write  $\mathfrak{s}(a, b, x) = \frac{\text{ReLU}(x-a) - \text{ReLU}(x-b)}{b-a}$ . Thus,  $|\mathcal{C}\text{-}\mathfrak{s}(m+1+c, a, b, x) - \mathfrak{s}(a, b, x)| \leq \frac{2 \cdot 2^{-m-1-c}}{b-a}$ , using the triangle inequality. Take  $c$  such that  $\frac{1}{b-a} \leq 2^c$ . ◀

The existence of the following function will play an important role to obtain the various functions of the next corollary.

► **Theorem 21.** *There exists some function  $\xi : \mathbb{N}^2 \rightarrow \mathbb{R}$  in  $\text{LDL}^\circ$  (illustrated in Figure 3) such that for all  $n, m \in \mathbb{N}$  and  $x \in [-2^n, 2^n]$ , whenever  $x \in [\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}]$ ,  $|\xi(2^m, 2^n, x) - \{x - \frac{1}{8}\}| \leq 2^{-m}$ .*

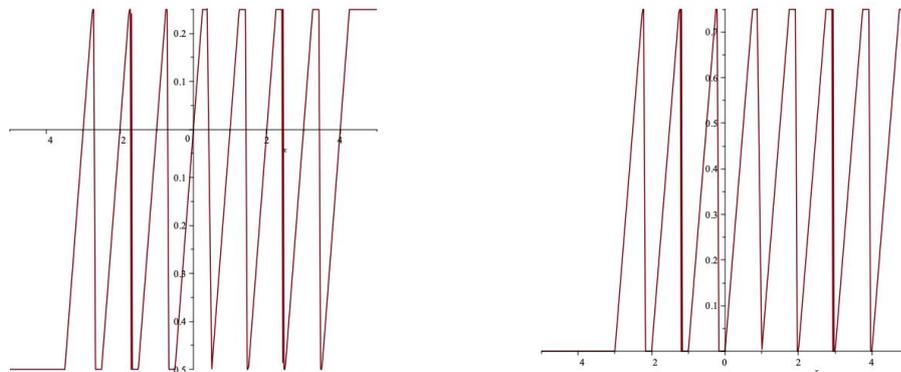


■ **Figure 3** Graphical representations of  $\xi(2, 4, x)$  obtained with maple: some details on the right.

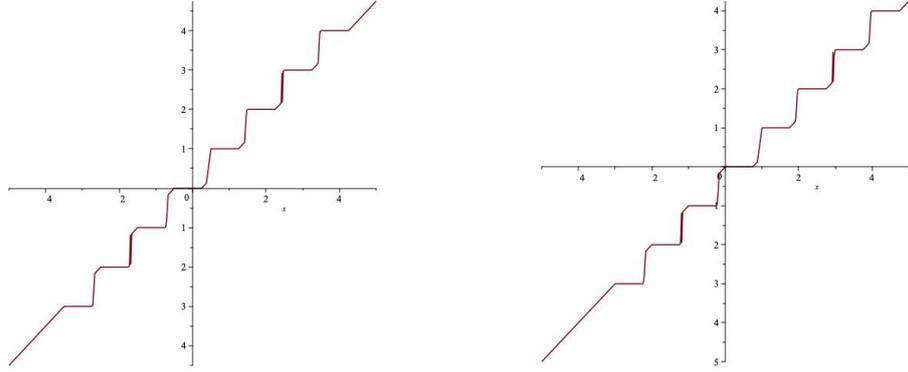
The main idea of the proof is, by parity, to reduce the problem to construct an auxiliary function  $\xi'$  that works for  $x \geq 0$ , writing  $\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} - \frac{3}{4} \mathcal{C}\text{-s}(2^{m+2}, 0, \frac{1}{8}, x)$ , and then proving that  $\xi'$  is definable in  $\text{LDL}^\circ$ , using an adhoc recursive (in  $n$ ) definition of it.

► **Corollary 22 (A bestiary of functions).** *There exist*

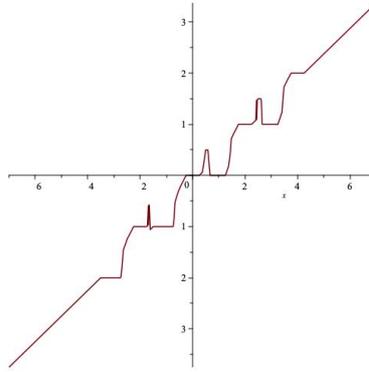
1.  $\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$  such that, for all  $n, m \in \mathbb{N}$ ,  $\lfloor x \rfloor \in [-2^n + 1, 2^n]$ , whenever  $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$ ,  $|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$ , and whenever  $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$ ,  $|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$  (see Figure 4).
2.  $\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$  such that, for all  $n, m \in \mathbb{N}$ ,  $\lfloor x \rfloor \in [-2^n + 1, 2^n]$ , whenever  $x \in [\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4}]$ ,  $\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor \leq 2^{-m}$ , and whenever  $x \in [\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}]$ ,  $|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$  (see Figure 5).
3.  $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \text{LDL}^\circ$  such that for all  $m, n \in \mathbb{N}$ ,  $\lfloor x \rfloor \in [-2^n + 1, 2^n]$ , whenever  $x \in [\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2}]$ ,  $|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m}$ , and whenever  $x \in [\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1]$ ,  $|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}$ .
4.  $\text{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \text{LDL}^\circ$  such that for all  $m, n \in \mathbb{N}$ ,  $\lfloor x \rfloor \in [-2^n + 1, 2^n]$ , whenever  $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$ ,  $|\text{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}$ .



■ **Figure 4** Graphical representation of  $\xi_1(2, 4, x)$  and  $\xi_2(2, 4, x)$  obtained with maple.



■ **Figure 5** Graphical representation of  $\sigma_1(2, 4, x)$  and  $\sigma_2(2, 4, x)$  obtained with maple.



■ **Figure 6** Graphical representation of  $\div_2(2, 4, x)$  obtained with maple.

5.  $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \text{LDL}^\circ$  such that for all  $m, n \in \mathbb{N}$ ,  $\lfloor x \rfloor \in [-2^n + 1, 2^n]$ , whenever  $x \in [\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}]$ ,  $|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m}$ , with  $//$  the integer division (see Figure 6).

**Proof.** Take  $\xi_1(M, N, x) = \xi(M, N, x - \frac{3}{8}) - \frac{1}{2}$ ,  $\xi_2(M, N, x) = \xi(N, x - \frac{7}{8})$ ,  $\sigma_i(M, N, x) = x - \xi_i(M, N, x)$ ,  $\lambda(M, N, x) = \mathcal{C}\text{-}\mathfrak{s}(2M, 1/4, 1/2, \xi(2M, N, x - 9/8))$ ,  $\text{mod}_2(M, N, x) = 1 - \lambda(M, N/2, \frac{1}{2}x + \frac{7}{8})$ ,  $\div_2(M, N, x) = \frac{1}{2}(\sigma_1(M, N, x) - \text{mod}_2(M, N, x))$ . ◀

Observing that for  $\bar{\text{if}}(d, l) = 4 \mathfrak{s}(1, 2, 1/2 + d + l/4) - 2$ , for  $l \in [0, 1]$ , we have  $\bar{\text{if}}(0, l) = 0$ , and  $\bar{\text{if}}(1, l) = l$ , and using Lemma 20 on this sigmoid, we get:

► **Lemma 23.** *There exists  $\mathcal{C}\text{-if} \in \text{LDL}^\circ$  such that,  $l \in [0, 1]$ , if we take  $|d' - 0| \leq 1/4$ , then  $|\mathcal{C}\text{-if}(d', l) - 0| \leq 2^{-m}$ , and if we take  $|d' - 1| \leq 1/4$ , then  $|\mathcal{C}\text{-if}(d', l) - l| \leq 2^{-m}$ .*

► **Lemma 24.** *Let  $\alpha_1, \alpha_2, \dots, \alpha_n$  be some integers, and  $V_1, V_2, \dots, V_n$  some constants. We write  $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$  for the function that maps any  $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$  to  $V_i$ , for all  $i \in \{1, \dots, n\}$ .*

*There is some function in  $\text{LDL}^\circ$ , that we write  $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$ , that maps any  $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$  to a real at distance at most  $2^{-m}$  of  $V_i$ , for all  $i \in \{1, \dots, n\}$ .*

► **Lemma 25.** *Let  $N$  be some integer. Let  $\alpha_1, \alpha_2, \dots, \alpha_n$  be some integers, and  $V_{i,j}$  for  $1 \leq i \leq n$  some constants, with  $0 \leq j < N$ . We write  $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$  for the function that maps any  $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$  and  $y \in [j - 1/4, j + 1/4]$  to  $V_{i,j}$ , for all  $i \in \{1, \dots, n\}$ ,  $j \in \{0, \dots, N-1\}$ .*

There is some function in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ , that we write  $\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$ , that maps any  $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$  and  $y \in [j - 1/4, j + 1/4]$  to a real at distance at most  $2^{-m}$  of  $V_{i,j}$ , for all  $i \in \{1, \dots, n\}$ ,  $j \in \{0, \dots, N-1\}$ .

#### 4 Simulating Turing machines with functions of $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$

This section is devoted to the simulation of a Turing machine using some analytic functions, and in particular functions from  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ . We use some ideas from [3] but with several improvements, as we need to deal with errors and avoid multiplications.

Consider without loss of generality some Turing machine  $M = (Q, \{0, 1, 3\}, q_{init}, \delta, F)$  using the symbols 0, 1, 3, where  $B = 0$  is the blank symbol.

► **Remark 26.** The reason of the choice of symbols 1 and 3 will be made clear later.

We assume  $Q = \{0, 1, \dots, |Q| - 1\}$ . Let  $\dots l_{-k}l_{-k+1} \dots l_{-1}l_0r_0r_1 \dots r_n \dots$  denote the content of the tape of the Turing machine  $M$ . In this representation, the head is in front of symbol  $r_0$ , and  $l_i, r_i \in \{0, 1, 3\}$  for all  $i$ . Such a configuration  $C$  can be denoted by  $C = (q, l, r)$ , where  $l, r \in \Sigma^\omega$  are words over alphabet  $\Sigma = \{0, 1, 3\}$  and  $q \in Q$  denotes the internal state of  $M$ . Write:  $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$  for the function that maps a word  $w = w_0w_1w_2 \dots$  to the dyadic  $\gamma_{word}(w) = \sum_{n \geq 0} w_n 4^{-(n+1)}$ .

The idea is that such a configuration  $C$  can also be encoded by some element  $\bar{C} = (q, \bar{l}, \bar{r}) \in \mathbb{N} \times \mathbb{R}^2$ , by considering  $\bar{r} = \gamma_{word}(r)$  and  $\bar{l} = \gamma_{word}(l)$ . In other words, we encode the configuration of a bi-infinite tape Turing machine  $M$  by real numbers using their radix 4 encoding, but using only digits 1,3. Notice that this lives in  $Q \times [0, 1]^2$ . Denoting the image of  $\gamma_{word} : \Sigma^\omega \rightarrow \mathbb{R}$  by  $\mathcal{I}$ , this even lives in  $Q \times \mathcal{I}^2$ .

► **Remark 27.** Notice that  $\mathcal{I}$  is a Cantor-like set: it corresponds to the rational numbers that can be written using only 1 and 3 in base 4. We write  $\mathcal{I}_S$  for those with at most  $S$  digits after the point (i.e. of the form  $n/4^S$  for some integer  $n$ ).

► **Lemma 28.** We can construct some function  $\overline{Next}$  in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$  that simulates one step of  $M$ : given a configuration  $C$ , writing  $C'$  for the next configuration, we have for all integer  $m$ ,  $\|\overline{Next}(2^m, \bar{C}) - \bar{C}'\| \leq 2^{-m}$ .

**Proof.** We can write  $l = l_0l^\bullet$  and  $r = r_0r^\bullet$ , where  $l_0$  and  $r_0$  are the first letters of  $l$  and  $r$ , and  $l^\bullet$  and  $r^\bullet$  corresponding to the (possibly infinite) word  $l_{-1}l_{-2} \dots$  and  $r_1r_2 \dots$  respectively.

$$\begin{array}{ccccccc} \dots & l^\bullet & | & l_0 & | & r_0 & | & r^\bullet & \dots \\ \hline & & & \underbrace{\hspace{2cm}} & & \underbrace{\hspace{2cm}} & & & \\ & & & l & & r & & & \end{array}$$

The function  $Next$  is of the form  $Next(q, l, r) = Next(q, l^\bullet l_0, r_0 r^\bullet) = (q', l', r')$  defined as a definition by case of type:

$$(q', l', r') = \begin{cases} (q', l^\bullet l_0 x, r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', l^\bullet, l_0 x r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}$$

This can be rewritten as a first candidate for the function  $\overline{Next}$ . Consider the similar function working over the representation of the configurations as reals, considering  $r_0 = \lfloor 4\bar{r} \rfloor$

$$\begin{aligned} \overline{Next}(q, \bar{l}, \bar{r}) &= \overline{Next}(q, \bar{l}^\bullet \bar{l}_0, \bar{r}_0 \bar{r}^\bullet) = (q', \bar{l}', \bar{r}') \\ &= \begin{cases} (q', \bar{l}^\bullet \bar{l}_0 x, \bar{r}^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', \bar{l}^\bullet, \bar{l}_0 x \bar{r}^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases} \end{aligned}$$

- in the first case “ $\rightarrow$ ” :  $\bar{l} = 4^{-1}\bar{l} + 4^{-1}x$  and  $\bar{r}' = \bar{r}^\bullet = \{4\bar{r}\}$
  - in the second case “ $\leftarrow$ ” :  $\bar{l} = \bar{l}^\bullet = \{4\bar{l}\}$  and  $\bar{r}' = 4^{-2}\{4\bar{r}\} + 4^{-2}x + \lfloor 4\bar{l} \rfloor / 4$
- (6)

We introduce the following functions:  $\rightarrow: Q \times \{0, 1, 3\} \mapsto \{0, 1\}$  and  $\leftarrow: Q \times \{0, 1, 3\} \mapsto \{0, 1\}$  such that  $\rightarrow(q, a)$  (respectively:  $\leftarrow(q, a)$ ) is 1 when  $\delta(q, a) = (\cdot, \cdot, \rightarrow)$  (resp.  $(\cdot, \cdot, \leftarrow)$ ), i.e. the head moves right (resp. left), and 0 otherwise. We define  $nextq_a^q = q'$  if  $\delta(q, a) = (q', \cdot, \cdot)$ , i.e. values  $(q', x, m)$  for some  $x$  and  $m \in \{\leftarrow, \rightarrow\}$ .

We can rewrite  $\overline{Next}(q, \bar{l}, \bar{r}) = (q', \bar{l}', \bar{r}')$  as  $\bar{l}' = \sum_{q, r_0} \left[ \rightarrow(q, r_0) \left( \frac{\bar{l}}{4} + \frac{x}{4} \right) + \leftarrow(q, r_0) \{4\bar{l}\} \right]$  and  $\bar{r}' = \sum_{q, r_0} \left[ \rightarrow(q, r_0) \{4\bar{r}\} + \leftarrow(q, r_0) \left( \frac{\{4r\}}{4^2} + \frac{x}{4^2} + \frac{\lfloor 4\bar{l} \rfloor}{4} \right) \right]$ , and, using notation of Lemma 25,  $q' = \text{send}((q, r) \mapsto nextq_a^q)_{q \in Q, r \in \{0, 1, 3\}}(q, \lfloor 4\bar{r} \rfloor)$ .

Our problem with such expressions is that they involve some discontinuous functions such as the integer part and the fractional part function, and we would rather have analytic (hence continuous) functions. A key point is that from our trick of using only symbols 1 and 3, we are sure that in an expression like  $\lfloor 4\bar{r} \rfloor$ , either it values 0 (this is the specific case where there remain only blanks in  $r$ ), or that  $4\bar{r}$  lives in an interval  $[1, 2]$  or in interval  $[3, 4]$ . That means that we could replace  $\lfloor 4\bar{r} \rfloor$  by  $\sigma(4\bar{r})$  if we take  $\sigma$  as some continuous function that would be affine and values respectively 0, 1 and 3 on  $\{0\} \cup [1, 2] \cup [3, 4]$  (that is to say matches  $\lfloor 4\bar{r} \rfloor$  on this domain). A possible candidate is  $\sigma(x) = \mathfrak{s}(1/4, 3/4, x) + \mathfrak{s}(9/4, 11/4, x)$ . Then considering  $\xi(x) = x - \sigma(x)$ , then  $\xi(4\bar{r})$  would be the same as  $\{4\bar{r}\}$ : that is, considering  $r_0 = \sigma(4\bar{r})$ , replacing in the above expression every  $\{4\cdot\}$  by  $\xi(\cdot)$ , and every  $\lfloor \cdot \rfloor$  by  $\sigma(\cdot)$ , and get something that would still work the same, but using only continuous functions.

But, we would like to go to some analytic functions and not only continuous functions, and it is well-known that an analytic function that equals some affine function on some interval (e.g. on  $[1, 2]$ ) must be affine, and hence cannot be 3 on  $[3, 4]$ . But the point is that we can try to tolerate errors, and replace  $\mathfrak{s}(\cdot, \cdot)$  by  $\mathcal{C}\text{-}\mathfrak{s}(2^{m+c}, \cdot, \cdot)$  in the expressions above for  $\sigma$  and  $\xi$ , taking  $c$  such that  $(3 + 1/4^2)3|Q| \leq 2^c$ . This would just introduce some error at most  $(3 + 1/4^2)3|Q|2^{-c}2^{-m} \leq 2^{-m}$ .

► **Remark 29.** We could also replace every  $\rightarrow(q, r)$  in above expressions for  $\bar{l}'$  and  $\bar{r}'$  by  $\mathcal{C}\text{-}\text{send}(k, (q, r) \mapsto \rightarrow(q, r))(q, \sigma(4\bar{r}))$ , for a suitable error bound  $k$ , and symmetrically for  $\leftarrow(q, r)$ . However, if we do so, we still might have some multiplications in the above expressions.

The key is to use Lemma 23: we can also write the above expressions as

$$\begin{aligned} \bar{l}' &= \sum_{q, r} \left[ \mathcal{C}\text{-if} \left( 2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \rightarrow(q, r))(q, \sigma(4\bar{r})), \frac{\bar{l}}{4} + \frac{x}{4} \right) \right. \\ &\quad \left. + \mathcal{C}\text{-if} \left( 2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \leftarrow(q, r))(q, \sigma(4\bar{r})), \xi(4\bar{l}) \right) \right] \\ \bar{r}' &= \sum_{q, r} \left[ \mathcal{C}\text{-if} \left( 2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \rightarrow(q, r))(q, \sigma(4\bar{r})), \xi(4\bar{r}) \right) \right. \\ &\quad \left. + \mathcal{C}\text{-if} \left( 2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \leftarrow(q, r))(q, \sigma(4\bar{r})), \frac{\xi(4r)}{4^2} + \frac{x}{4^2} + \frac{\sigma(4\bar{l})}{4} \right) \right] \end{aligned}$$

and still have the same bound on the error. ◀

Once we have one step, we would like to simulate some arbitrary computation of a Turing machine, by considering the iterations of function *Next*.

The problem of above construction, is that, even if we start from the exact encoding  $\bar{C}$  of a configuration, it introduces some error (even if at most  $2^{-m}$ ). If we want to apply again the function *Next*, then we will start not exactly from the encoding of a configuration.

Looking at the choice of the function  $\sigma$ , a small error can be tolerated (roughly if the process does not involve points at distance less than  $1/4$  of  $\mathcal{I}$ ), but this error is amplified (roughly multiplied by 4 on some component), before introducing some new errors (even if at most  $2^{-m}$ ). The point is that if we repeat the process, very soon it will be amplified, up to a level where we have no true idea or control about what becomes the value of above function.

However, if we know some bound on the space used by the Turing machine, we can correct it to get at most some fixed additive error: a Turing machine using a space  $S$  uses at most  $S$  cells to the right and to the left of the initial position of its head. Consequently, a configuration  $C = (q, l, r)$  of such a machine involves words  $l$  and  $r$  of length at most  $S$ . Their encoding  $\bar{l}$ , and  $\bar{r}$  are expected to remain in  $\mathcal{I}_{S+1}$ . Consider  $\text{round}_{S+1}(\bar{l}) = \lfloor 4^{S+1}\bar{l} \rfloor / 4^{S+1}$ . For a point  $\bar{l}$  of  $\mathcal{I}_{S+1}$ ,  $4^{S+1}\bar{l}$  is an integer, and  $\bar{l} = \text{round}_{S+1}(\bar{l})$ . But now, for a point  $\tilde{\bar{l}}$  at distance less than  $4^{-(S+2)}$  from a point  $\bar{l} \in \mathcal{I}_{S+1}$ ,  $\text{round}_{S+1}(\tilde{\bar{l}}) = \bar{l}$ . In other words,  $\text{round}_{S+1}$  “deletes” errors of order  $4^{-(S+2)}$ . Consequently, we can replace every  $\bar{l}$  in above expressions by  $\sigma_1(2^{2S+4}, 2^{2S+3}, 4^{S+1}\bar{l})/4^{S+1}$ , as this is close to  $\text{round}_{S+1}(\bar{l})$ , and the same for  $\bar{r}$ , where  $\sigma_1$  is the function from Corollary 22. We could also replace  $m$  by  $m + 2S + 4$  to guarantee that  $2^{-m} \leq 4^{-(S+2)}$ . We get the following important improvement of the previous lemma:

► **Lemma 30.** *We can construct some function  $\overline{\text{Next}}$  in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$  that simulates one step of  $M$ , i.e. that computes the Next function sending a configuration  $\bar{C}$  of Turing machine  $M$  to  $\bar{C}'$ , where  $C'$  is the next one:  $\|\text{Next}(2^m, 2^S, \bar{C}) - \bar{C}'\| \leq 2^{-m}$ . Furthermore, it is robust to errors on its input, up to space  $S$ : considering  $\|\tilde{C} - \bar{C}\| \leq 4^{-(S+2)}$ ,  $\|\text{Next}(2^m, 2^S, \tilde{C}) - \bar{C}'\| \leq 2^{-m}$  remains true.*

► **Proposition 31.** *Consider some Turing machine  $M$  that computes some function  $f : \Sigma^* \rightarrow \Sigma^*$  in some time  $T(\ell(\omega))$  on input  $\omega$ . One can construct some function  $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$  in  $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$  that does the same:  $\tilde{f}(2^m, 2^{T(\ell(\omega))}, \gamma_{\text{word}}(\omega))$  that is at most  $2^{-m}$  far from  $\gamma_{\text{word}}(f(\omega))$ .*

**Proof.** The idea is to define the function  $\overline{\text{Exec}}$  that maps some time  $2^t$  and some initial configuration  $C$  to the configuration at time  $t$ . This can be obtained using previous lemma by  $\overline{\text{Exec}}(2^m, 0, 2^T, C) = C$  and  $\overline{\text{Exec}}(2^m, 2^{t+1}, 2^T, C) = \overline{\text{Next}}(2^m, 2^T, \overline{\text{Exec}}(2^m, 2^t, 2^T, C))$ .

We can then get the value of the computation as  $\overline{\text{Exec}}(2^m, 2^{T(\ell(\omega))}, 2^{T(\ell(\omega))}, C_{\text{init}})$  on input  $\omega$ , considering  $C_{\text{init}} = (q_0, 0, \gamma_{\text{word}}(\omega))$ . By applying some projection, we get the following function  $\tilde{f}(2^m, 2^T, y) = \pi_3^3(\overline{\text{Exec}}(2^m, 2^T, 2^T, (q_0, 0, y)))$  that satisfies the property. ◀

Actually, in order to get **FPSPACE**, observe that we can also replace the linear length ODE by a linear ODE.

► **Proposition 32.** *Consider some Turing machine  $M$  that computes some function  $f : \Sigma^* \rightarrow \Sigma^*$  in some polynomial space  $S(\ell(\omega))$  on input  $\omega$ . One can construct some function  $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$  in  $\mathbb{R}\mathbb{L}\mathbb{D}^\circ$  that does the same: we have  $\tilde{f}(2^m, 2^{S(\ell(\omega))}, \gamma_{\text{word}}(\omega))$  that is at most  $2^{-m}$  far from  $\gamma_{\text{word}}(f(\omega))$ .*

**Proof.** The idea is the same, but not working with powers of 2, and with linear ODE: define the function  $\overline{\text{Exec}}$  that maps some time  $t$  and some initial configuration  $C$  to the configuration at time  $t$ . This can be obtained using previous lemma by  $\overline{\text{Exec}}(2^m, 0, 2^S, C) = C$  and  $\overline{\text{Exec}}(2^m, t+1, 2^S, C) = \overline{\text{Next}}(2^m, 2^S, \overline{\text{Exec}}(2^m, t, 2^S, C))$ .

In order to claim this is a robust linear ODE, we need to state that  $\text{Exec}(2^m, t, 2^S, C)$  is polynomially numerically stable: but this holds, since to estimate this value at  $2^{-n}$  it is sufficient to work at precision  $4^{-\max(m, n, S+2)}$  (independently of  $t$ , from the rounding).

We can then get the value of the computation as  $\overline{\text{Exec}}(2^m, 2^{S(\ell(\omega))}, 2^{S(\ell(\omega))}, C_{\text{init}})$  on input  $\omega$ , considering  $C_{\text{init}} = (q_0, 0, \gamma_{\text{word}}(\omega))$ . By applying some projection, we get the following function  $\tilde{f}(2^m, 2^S, y) = \pi_3^3(\overline{\text{Exec}}(2^m, S, 2^S, (q_0, 0, y)))$  that satisfies the property. ◀

## 5 Converting integers and dyadics to words, and conversely

One point of our simulations of Turing machines is that they work over  $\mathcal{I}$ , through encoding  $\gamma_{word}$ , while we would like to talk about integers and real numbers: we need to be able to convert an integer (more generally a dyadic) into some encoding over  $\mathcal{I}$  and conversely.

Fix the following encoding: every digit in the binary expansion of  $d$  is encoded by a pair of symbols in the radix 4 expansion of  $\bar{d} \in \mathcal{I} \cap [0, 1]$ : digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the “decimal” point in  $d$ , and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for  $d = 101.1$  in base 2,  $\bar{d} = 0.13111333$  in base 4.

By iterating  $\ell(n)$  times the function

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} (\div_2(\bar{r}_1), (\bar{l}_2 + 5)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 0 \\ (\div_2(\bar{r}_1), (\bar{l}_2 + 7)/4) & \text{whenever } \text{mod}_2(\bar{r}_1) = 1. \end{cases}$$

over  $(n, 0)$ , and then projecting on the second argument, we can prove:

► **Lemma 33** (From  $\mathbb{N}$  to  $\mathcal{I}$ ). *We can construct some function  $Decode : \mathbb{N}^2 \rightarrow \mathbb{R}$  in  $\text{LDL}^\circ$  that maps  $m$  and  $n$  to some point at distance less than  $2^{-m}$  from  $\gamma_{word}(\bar{n})$ .*

This technique can be extended to consider decoding of tuples: there is a function  $Decode : \mathbb{N}^{d+1} \rightarrow \mathbb{R}$  in  $\text{LDL}^\circ$  that maps  $m$  and  $\mathbf{n}$  to some point at distance less than  $2^{-m}$  from  $\gamma_{word}(\bar{\mathbf{n}})$ , with  $\bar{\mathbf{n}}$  defined componentwise.

Conversely, given  $\bar{d}$ , we need a way to construct  $d$ . Actually, as we will need to avoid multiplications, we state that we can even do something stronger: given  $\bar{d}$ , and (some bounded)  $\lambda$  we can construct  $\lambda d$ .

► **Lemma 34** (From  $\mathcal{I}$  to  $\mathbb{R}$ , and multiplying in parallel). *We can construct some function  $EncodeMul : \mathbb{N}^2 \times [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}$  in  $\text{LDL}^\circ$  that maps  $m$ ,  $2^S$ ,  $\gamma_{word}(\bar{d})$  and (bounded)  $\lambda$  to some real at distance at most  $2^{-m}$  from  $\lambda d$ , whenever  $\bar{d}$  is of length less than  $S$ .*

## 6 Proofs and applications

When we say that a function  $f : S_1 \times \cdots \times S_d \rightarrow \mathbb{R}^d$  is (respectively: polynomial time or space) computable this will always be in the sense of computable analysis: see e.g. [7, 19]. We actually follow the formalisation in [3] of required concepts from computable analysis, able to mix complexity issues dealing with integer and real arguments. Theorem 8 follows from point 1. of next Proposition for one inclusion, and previous simulation of Turing machines for the other.

► **Proposition 35.**

1. *All functions of  $\text{LDL}^\circ$  are computable (in the sense of computable analysis) in polynomial time.*
2. *All functions of  $\text{RLD}$  are computable (in the sense of computable analysis) in polynomial space.*

The proof of the proposition consists in observing this holds for the basic functions and that composition preserves polynomial time (respectively: space) computability and also by linear length ODEs. This latter fact is established by computable analysis arguments, reasoning on some explicit formula giving the solution of linear length ODE. Regarding space, the main issue is the need to prove the schema given by Definition 18 guarantees  $\mathbf{f}$  is in **FSPACE**, when  $\mathbf{u}$ ,  $\mathbf{g}$ , and  $\mathbf{h}$  are. Assuming condition 1. of Definition 18 would not be

sufficient: the problem is  $\mathbf{f}(x, \mathbf{y})$  may polynomially grow too fast or have a modulus function that would grow too fast. The point is, in Definition 18, we assumed  $\mathbf{f}$  to be both bounded and satisfying **2.**, i.e. polynomial numerical robustness. With these hypotheses, it is sufficient to work with the precision given by this robustness condition and these conditions guarantee the validity of computing with such approximated values.

We now go to various applications of it and of our toolbox. First, we state a characterisation of **FPTIME** for general functions, covering both the case of a function  $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ ,  $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$  as a special case: only the first type (sequences) was covered by [3].

► **Theorem 36** (Theorem 9). *A function  $\mathbf{f} : \mathbb{R}^d \times \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d'}$  is computable in polynomial time iff there exists  $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''+2} \rightarrow \mathbb{R}^{d'} \in \text{LDL}^\circ$  such that for all  $\mathbf{x} \in \mathbb{R}^d$ ,  $X \in \mathbb{N}$ ,  $\mathbf{m} \in \mathbb{N}^{d''}$ ,  $n \in \mathbb{N}$ ,  $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$ .*

The reverse implication of Theorem 36 follows from Proposition 35, (1.) and arguments from computable analysis. For the direct implication, for sequences, that is to say, functions of type  $\mathbf{f} : \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d'}$  (i.e.  $d = 0$ , the case considered in [3]) we are almost done: reasoning componentwise, we only need to consider  $f : \mathbb{N}^{d''} \rightarrow \mathbb{R}$  (i.e.  $d' = 1$ ). As the function is polynomial time computable, this means that there is a polynomial time computable function  $g : \mathbb{N}^{d''+1} \rightarrow \{1, 3\}^*$  so that on  $\mathbf{m}, 2^n$ , it provides the encoding  $\overline{\phi(\mathbf{m}, n)}$  of some dyadic  $\phi(\mathbf{m}, n)$  with  $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$  for all  $\mathbf{m}$ . The problem is then to decode, compute and encode the result to produce this dyadic, using our previous toolbox.

More precisely, from Proposition 31, we get  $\tilde{g}$  with  $|\tilde{g}(2^e, 2^{p(\max(\mathbf{m}, n))}, \text{Decode}(2^e, \mathbf{m}, n)) - \gamma_{\text{word}}(g(\mathbf{m}, n))| \leq 2^{-e}$  for some polynomial  $p$  corresponding to the time required to compute  $g$ , and  $e = \max(p(\max(\mathbf{m}, n)), n)$ . Then we need to transform the value to the correct dyadic: we mean  $\tilde{\mathbf{f}}(\mathbf{m}, n) = \text{EncodeMul}(2^e, 2^t, \tilde{g}(2^e, 2^t, \text{Decode}(2^e, \mathbf{m}, n)), 1)$ , where  $t = p(\max(\mathbf{m}, n))$ ,  $e = \max(p(\max(\mathbf{m}, n)), n)$  provides a solution such that  $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ .

► **Remark 37.** This is basically what is done in [3], except that we do it here with analytic functions. However, as already observed in [3], this cannot be done for the case  $d \geq 1$ , i.e. for example for  $f : \mathbb{R} \rightarrow \mathbb{R}$ . The problem is that we used the fact that we can decode:  $\text{Decode}$  maps an integer  $n$  to its encoding  $\bar{n}$  (but is not guaranteed to do something valid on non-integers). There cannot exist such functions that would be valid over all reals, as such function must be continuous, and there is no way to map continuously real numbers to finite words. This is where the approach of the article [3] is stuck.

To solve this, we use an adaptive barycentric technique. By lack of space, we discuss only the case of a polynomial time computable function  $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ . From standard arguments from computable analysis (see e.g. [Corollary 2.21][13]), the following holds and the point is to be able to realise all this with functions from  $\text{LDL}^\circ$ .

► **Lemma 38.** *Assume  $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$  is computable in polynomial time. There exists some polynomial  $m : \mathbb{N}^2 \rightarrow \mathbb{N}$  and some  $\tilde{f} : \mathbb{N}^4 \rightarrow \mathbb{Z}$  computable in polynomial time such that for all  $x \in \mathbb{R}$ ,  $\|2^{-n} \tilde{f}(\lfloor 2^{m(n, M)} x \rfloor, u, 2^M, 2^n) - f(x, u)\| \leq 2^{-n}$  whenever  $\frac{x}{2^{m(n, M)}} \in [-2^M, 2^M]$ .*

Assume we consider an approximation  $\sigma_i$  (with either  $i = 1$  or  $i = 2$ ) of the integer part function given by Lemma 22. Then, given  $n, M$ , when  $2^{m(n, M)} x$  falls in some suitable interval  $I_i$  for  $\sigma_i$  (see statement of Lemma 22), we are sure that  $\sigma_i(2^e, 2^{m(n, M)+X+1}, 2^{m(n, M)} x)$  is at some distance upon control from  $\lfloor 2^{m(n, M)} x \rfloor$ . Consequently,  $2^{-n} \tilde{f}(\sigma_i(2^{m(n, M)+X+1}, 2^{m(n, M)} x), u, 2^M, 2^n)$  provides some  $2^{-n}$ -approximation of  $f(x, u)$ , up to some error upon control. When this holds, we then use an argument similar to what we describe for sequences: using functions from  $\text{LDL}^\circ$ , we can decode, compute, and encode the result to provide this dyadic. It is provided by an expression  $\text{Formula}_i(x, u, M, n)$  of the form  $\text{EncodeMul}(2^e, 2^t, \tilde{f}(2^2, 2^t, \text{Decode}(2^e, \sigma_i(2^e, 2^M, 2^{m(n, M)} x))), 2^{-n})$ .

The problem is that it might also be the case that  $2^{m(n,M)}x$  falls in the complement of the intervals  $(I_i)_i$ . In that case, we have no clear idea of what could be the value of  $\sigma_i(2^e, 2^{m(n,M)+X+1}, 2^{m(n,M)}x)$ , and consequently of what might be the value of the above expression  $Formula_i(x, u, M, n)$ . But the point is that when it happens for an  $x$  for  $\sigma_1$ , we could have used  $\sigma_2$ , and this would work, as one can check that the intervals of type  $I_1$  covers the complements of the intervals of type  $I_2$  and conversely. They also overlap, but when  $x$  is both in some  $I_1$  and  $I_2$ ,  $Formula_1(x, u, M, n)$  and  $Formula_2(x, u, M, n)$  may differ, but they are both  $2^{-n}$  approximation of  $f(x)$ .

The key is to compute some suitable ‘‘adaptive’’ barycenter, using function  $\lambda$ , provided by Corollary 22. Writing  $\approx$  for the fact that two values are closed up to some controlled bounded error, observe from the statements of Lemma 22 that whenever  $\lambda(\cdot, 2^n, x) \approx 0$ , we know that  $\sigma_2(\cdot, 2^n, x) \approx [x]$ ; whenever  $\lambda(\cdot, 2^n, x) \approx 1$  we know that  $\sigma_1(\cdot, 2^n, x) \approx [x]$ ; whenever  $\lambda(\cdot, 2^n, x) \in (0, 1)$ , we know that  $\sigma_1(\cdot, 2^n, x) \approx [x] + 1$  and  $\sigma_2(\cdot, 2^n, x) \approx [x]$ . That means that if we consider  $\lambda(\cdot, 2^n, x)Formula_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, x))Formula_2(x, u, M, n)$  we are sure to be close (up to some bounded error) to some  $2^{-n}$  approximation of  $f(x)$ . There remains that this requires some multiplication with  $\lambda$ . But from the form of  $Formula_i(x, u, M, n)$ , this could be also be written as follows, ending the proof of Theorem 36.

$$\begin{aligned} & EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_1(2^e, 2^M, 2^{m(n,M)}x))), \lambda(2^e, 2^M, 2^{m(n,M)}x)2^{-n}) + \\ & EncodeMul(2^e, 2^t, \tilde{f}(2^e, 2^t, Decode(2^e, \sigma_2(2^e, 2^M, 2^{m(n,M)}x))), (1 - \lambda(2^e, 2^M, 2^{m(n,M)}x))2^{-n}) \quad (7) \end{aligned}$$

From the fact that we have the reverse direction in Theorem 36, it is natural to consider the operation that maps  $\tilde{\mathbf{f}}$  to  $\mathbf{f}$ . Namely, we introduce the operation  $ELim$  ( $ELim$  stands for Effective Limit):

► **Definition 39** (Operation  $ELim$ ). *Given  $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''} \times \mathbb{N} \rightarrow \mathbb{R}^{d'} \in \mathbb{LDL}^\circ$  such that for all  $\mathbf{x} \in \mathbb{R}^d$ ,  $X \in \mathbb{N}$ ,  $\mathbf{x} \in [-2^X, 2^X]$ ,  $\mathbf{m} \in \mathbb{N}^{d''}$ ,  $n \in \mathbb{N}$ ,  $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$ , then  $ELim(\tilde{\mathbf{f}})$  is the (clearly uniquely defined) corresponding function  $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ .*

► **Theorem 40.** *A continuous function  $\mathbf{f}$  is computable in polynomial time if and only if all its components belong to  $\mathbb{LDL}^\circ$ , where  $\mathbb{LDL}^\circ = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \overline{cond}(x), \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE, } ELim]$ .*

For the reverse direction, by induction, the only thing to prove is that the class of functions from to the integers computable in polynomial time is preserved by the operation  $ELim$ . Take such a function  $\tilde{\mathbf{f}}$ . By definition, given  $\mathbf{x}, \mathbf{m}, X$  we can compute  $\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n)$  with precision  $2^{-n}$  in time polynomial in  $n$ . This must be, by definition of  $ELim$  schema, some approximation of  $\mathbf{f}(\mathbf{x}, \mathbf{m})$  over  $[-2^X, 2^X]$ , and hence  $\mathbf{f}$  is computable in polynomial time. This also gives directly Theorem 9 as a corollary.

We obtain the statements for polynomial space computability (Theorems 10 and 11) replacing  $\mathbb{LDL}^\circ$  by  $\mathbb{RLD}^\circ$ , using similar reasoning about space instead of time, considering point 2. instead of 1. of Proposition 35, and Proposition 32 instead of Proposition 31.

---

## References

- 1 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.
- 2 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. Submitted. Journal version of [3]. Preliminary version available on [arXiv:2209.13599](https://arxiv.org/abs/2209.13599).

- 3 Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality – 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 – September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022. MCU'22 Best Student Paper Award. doi:10.1007/978-3-031-13502-6\_4.
- 4 Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggenes, and Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPICs*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 5 Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023.
- 6 Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In *International Colloquium on Automata Language Programming, ICALP'2017*, 2017.
- 7 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.
- 8 P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998.
- 9 Peter Clote and Evangelos Kranakis. *Boolean functions and computation models*. Springer Science & Business Media, 2013.
- 10 Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.
- 11 Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology-CMSB 2017*, 2017. CMSB'2017 Best Paper Award.
- 12 Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*, chapter Computability of Differential Equations, pages 71–99. Springer, 2018.
- 13 Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.
- 14 D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, number 960 in *Lecture Notes in Computer Science*, pages 177–194, 1995.
- 15 Daniel Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.
- 16 Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, 1993.
- 17 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer.
- 18 David B Thompson. Subrecursiveness: Machine-independent notions of computability in restricted time and storage. *Mathematical Systems Theory*, 6(1-2):3–15, 1972.
- 19 Klaus Weihrauch. *Computable Analysis: an Introduction*. Springer, 2000.



# MaxCut Above Guarantee

Ivan Bliznets  

Utrecht University, The Netherlands

Vladislav Epifanov 

HSE University, St. Petersburg, Russia

---

## Abstract

In this paper, we study the computational complexity of the Maximum Cut problem parameterized above guarantee. Our main result provides a linear kernel for the Maximum Cut problem in connected graphs parameterized above the spanning tree. This kernel significantly improves the previous  $O(k^5)$  kernel given by Madathil, Saurabh, and Zehavi [ToCS 2020]. We also provide subexponential running time algorithms for this problem in special classes of graphs: chordal, split, and co-bipartite. We complete the picture by lower bounds under the assumption of the ETH. Moreover, we initiate a study of the Maximum Cut problem above  $\frac{2}{3}|E|$  lower bound in tripartite graphs.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

**Keywords and phrases** Tripartite, 3-colorable, chordal, maximum cut, FPT-algorithm, linear kernel

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.22

**Funding** Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 853234).

**Acknowledgements** We want to thank anonymous reviewers for their suggestions that helped to improve the presentation of the paper.

## 1 Introduction

The maximum cut problem, MAX-CUT for short, is a well-studied optimization problem in computer science and graph theory. In this problem, we need to split the vertices of a given graph into two parts, such that the sum of weights of edges going between two parts is maximum. The weighted version of the decision problem was one of Karp’s 21 NP-complete problems [14]. However, it is known that even unweighted MAXIMUM CUT is NP-complete [10]. It is easy to see that the problem admits an FPT-algorithm parameterized by the solution size, since the size of solution is relatively large. Specifically, if  $m$  is the number of edges in the input graph, then using a greedy argument we can construct a cut of size  $\frac{m}{2}$ . That is why it is natural to study parameterization above lower bounds. Under such parameterization [13], our goal is to find a solution of size  $k + l(I)$  where  $I$  is the input instance,  $l(I)$  is a polynomial time computable lower bound and  $k$  is our parameter. For more details about parameterizations of this type, we refer interested readers to a very nice survey written by Gutin and Mnich [13]. Mahajan and Raman were the first who studied the MAXIMUM CUT problem parameterized above lower bounds [16]. They showed that MAXIMUM CUT parameterized above lower bound  $\frac{m}{2}$  admits an FPT-algorithm. Later Crowston et al. [4, 5] presented a  $2^{O(k)}$  algorithm and a kernel with  $O(k^5)$  vertices for MAXIMUM CUT in the connected graph under more refined lower bound  $\frac{m}{2} + \frac{n-1}{4}$ . Subsequently, Crowston et al. [3] presented an  $O(k^3)$ -vertex kernel, and finally, Etscheid and Mnich [8] constructed an  $O(k)$ -vertex kernel. However, MAXIMUM CUT admits other lower bounds. One such lower bounds for a connected graph is  $n - 1$ . To construct a cut of this size, it is enough to find a spanning tree and then partition vertices such that all edges of the



© Ivan Bliznets and Vladislav Epifanov;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 22; pp. 22:1–22:14

Leibniz International Proceedings in Informatics



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

tree appear in the cut. Note that lower bounds  $\frac{m}{2} + \frac{n-1}{4}$ ,  $n-1$  are incomparable. We denote MAXIMUM CUT parameterized above spanning tree by MAX-CUT-AST. Madathil et al. [15] presented an  $\mathcal{O}^*(8^k)$ -time algorithm and a kernel with  $O(k^5)$  vertices for MAX-CUT-AST. Note that the lower bounds  $\frac{m}{2} + \frac{n-1}{4}$  and  $n-1$  are incomparable, since in very sparse graphs  $n-1 > \frac{m}{2} + \frac{n-1}{4}$  but in dense graphs  $\frac{m}{2} + \frac{n-1}{4} > n-1$ . So, the linear kernel designed by Etscheid and Mnich [9] does not provide a linear kernel for the parameterization above  $n-1$ .

In this paper, we continue the study of MAX-CUT-AST. Our main result establishes an  $O(k)$ -vertex kernel for this problem. This result resolves an Open Problem 2 from the survey [13] and significantly improves upon the previous  $O(k^5)$ -vertex kernel.

► **Theorem 1.** *MAX-CUT-AST has a kernel with  $\mathcal{O}(k)$  vertices and  $\mathcal{O}(k)$  edges.*

► **Theorem 2.** *There is no kernel with sublinear number of vertices for MAX-CUT-AST unless ETH fails.*

MAXIMUM CUT does not admit a  $2^{o(m)}$ -time algorithm unless the ETH fails [2]. Hence, MAX-CUT-AST also does not admit  $2^{o(k)}$ -time algorithm unless the ETH fails. However, we show that we can get a significant improvement if we restrict our input to special classes of graphs. Specifically, we proved the following theorems.

► **Theorem 3.** *MAX-CUT-AST can be solved in a connected co-bipartite graph  $G$  in  $2^{\mathcal{O}(\sqrt{k})}$  poly time.*

► **Theorem 4.** *MAX-CUT-AST on connected chordal graphs and split graphs admits a  $2^{\mathcal{O}(\sqrt{k})}$  poly-time algorithm.*

We complement these results with conditional lower bounds assuming the Exponential Time Hypothesis.

► **Theorem 5.** *MAX-CUT-AST does not admit  $2^{o(\sqrt{k})}$  poly or  $2^{o(\sqrt{n})}$  poly algorithms on split graphs and on chordal graphs unless the ETH is false.*

► **Theorem 6.** *MAX-CUT-AST does not admit  $2^{o(\sqrt[4]{k})}$  poly algorithm on co-bipartite graphs unless the ETH is false.*

We also initiate a new parameterization of MAXIMUM CUT on tripartite graphs. First, we show that MAX-CUT-AST on tripartite graphs is essentially as hard as MAX-CUT-AST on general graphs. Note that it is easy to see that a tripartite graph admits cut of size  $\frac{2}{3}m$  where  $m$  is the number of edges. However, it is not clear how to find such cut if the partition is not provided in the input. We prove that we can construct a randomized polynomial time algorithm that finds such cut. In order to design the algorithm we employ semidefinite programming and technique used to find a best known approximation of MAXIMUM CUT [12].

► **Theorem 7.** *There is a polynomial time randomized algorithm that finds an edge cut of size  $\frac{2}{3}|E(G)|$  on an input tripartite graph  $G$ .*

Note that the lower bound  $\frac{2}{3}m$  is essentially tight since disjoint union of  $k$  triangles is a tripartite graph and the maximum cut equals to  $2k$  while the total number of edges is  $3k$ . Therefore it is natural to ask whether MAXIMUM CUT above  $\frac{2}{3}|E(G)|$  admits an FPT algorithm in tripartite graphs. Unfortunately, we have not resolved the question and state it as an open problem. However, we show that finding cuts significantly bigger than  $\frac{2}{3}|E(G)|$  is hard even in tripartite graphs that contain such cuts. Formally, we proved the following theorem.

► **Theorem 8.** *Assuming  $P \neq NP$  for any  $0 < \varepsilon < \frac{1}{6}$  there is no polynomial time algorithm that finds cut of size  $\geq (\frac{2}{3} + \varepsilon)|E|$  in tripartite graph if such cut exists.*

## 2 Preliminaries

In this paper we work with undirected simple graphs. Only in very special cases we allow multigraphs and multi-edges. We explicitly mention situation when we start working with multi-edges. We use standard graph notations which can be found in [7]. As usual we denote by  $n$  or  $|V(G)|$  number of vertices in a graph  $G$ . Similarly by  $m$  or  $|E(G)|$  we denote number of edges in a graph  $G$ . Generally by  $k$  we denote value of parameter that is under consideration at the moment of discussion. Complement of the graph  $G$  we denote by  $\bar{G}$ . We recall that  $V(\bar{G}) = V(G)$  and  $e \in E(\bar{G})$  if and only if  $e \notin E(G)$ . Recall that in the MAXIMUM CUT problem we need to partition vertices of a given graph into two parts such the number of edges between two parts is the maximum possible. By  $E_G(A, B)$  we denote the set of edges going from the set  $A$  to the set  $B$  in the graph  $G$ . If the graph  $G$  is clear from the context we omit subscript  $G$ . Partition of vertices of graph into two disjoint parts  $A$  and  $B$  we denote by  $(A, B)$ . Slightly abusing notation in some cases we refer to cut as a set of edges  $E(A, B)$  and in other cases we refer to cut as a partition of vertices  $(A, B)$ .  $\text{mc}(G)$  denotes size of the maximum cut of the graph  $G$ . We call a vertex an *articulation point/vertex* if its removal increase number of connected components. Recall that *block* is a maximal 2-connected component.

*Chordal* graphs are graphs that do not contain induced cycles on  $\ell$  vertices for  $\ell \geq 4$ . *Co-bipartite* graphs are graphs whose complement graph is bipartite.

Many of our results concern the parameterized complexity of the problems, including fixed-parameter tractable algorithms, kernelization algorithms, and some hardness results for certain parameters. For a detailed survey in parameterized algorithms we refer to the book of Cygan et al [6]. Due to the space constraints, proofs of lemmas marked by  $(\star)$  are omitted.

## 3 Kernel

In this section we provide a linear kernel for MAX-CUT-AST. Recall that in this problem, we are given a connected graph  $G$ , integer  $k$ , and our goal is to determine if  $G$  has a maximum cut of size at least  $n - 1 + k$ . Before we proceed we formulate lemma by Mathadil et al. [15] that we are using in our algorithm.

► **Lemma 9** ([15]). *There exists a polynomial time algorithm that for  $(G, k)$  either concludes that graph  $G$  has a cut of size at least  $|V(G)| - 1 + k$  or finds a set  $S \subseteq V(G)$  such that  $|S| \leq 3k$  and each block of  $G - S$  is either a clique or a cycle.*

► **Theorem 1.** *MAX-CUT-AST has a kernel with  $\mathcal{O}(k)$  vertices and  $\mathcal{O}(k)$  edges.*

**Proof.** First of all we apply algorithm from Lemma 9. After this step either we conclude that input instance is Yes-instance and we can output a trivial kernel or we find a set  $S$  with properties described in Lemma 9. In the later case we proceed with an application of reduction rules. Our reduction rules maintain the following properties: (i) graph is connected; (ii) each block of  $G - S$  is either a cycle or a clique, however some of these cycles might become cycles of length 2 (our graph might become a multi-graph).

We call a vertex of  $G - S$  *special* if it has an edge to  $S$  or if it is an articulation point of  $G - S$ .

We are ready to present our first reduction rule.

<b>Rule 1:</b>	Let $U$ be a block in $G \setminus S$ , $v \in U$ is the only special vertex in $U$ and $U \neq \{v\}$
Remove:	Vertices $U \setminus \{v\}$
Parameter:	Reduce $k$ by $\text{mc}(U) - ( U  - 1)$

## 22:4 MaxCut Above Guarantee

**Proof of correctness.** Note that  $U$  has no other articulation points or vertices with edges to  $S$  except  $v$ . Therefore,  $v$  is an articulation point of the whole graph  $G$ . Hence, the maximum cut in  $G$  equals to the sum of the cut in  $U$  and  $G - (U \setminus \{v\})$ .

This reduction deletes all edges in  $U$ , so the maximum cut decreases by  $\text{mc}(U)$ . Besides we delete  $|U| - 1$  vertices. Hence, the parameter should be decreased by  $\text{mc}(U) - (|U| - 1)$ . ◀

**Rule 2:** Let  $v$  has no neighbors in  $G \setminus S$ , and exactly one neighbor in  $S$  i.e.  $N(v) = u \in S$   
 Remove:  $v$   
 Parameter: Reduce  $k$  by multiplicity of edge  $uv - 1$

**Proof of correctness.** Under this constrains  $v$  is either a pendant vertex in  $G$  or vertex connected with some other vertex  $u$  by multiple edges. Removal of  $v$  decreases the maximum cut by multiplicity of edge  $uv$  and number of vertices decreases by one. ◀

Since a block can consist of one vertex only in case this vertex is isolated, exhaustive application of these two rules leave every block with at least two special vertices or the whole connected component is a single vertex with at least two edges towards  $S$  and neighborhood of size at least 2.

► **Lemma 11.** *In a graph where Rules 1, 2 do not apply, every connected component of  $G - S$  has at least two edges to  $S$ .*

**Proof.** Let  $C$  be a connected component of  $G - S$  such that  $|E(S, C)| = 1$ . Therefore, at most one vertex from  $C$  is a special vertex but not an articulation point.

If  $C$  is composed of a single block then it has no articulation points and therefore has a block with only one special vertex. Hence, either Rule 1 or Rule 2 must be applicable.

Otherwise  $C$  has at least two leaf blocks, each of them having exactly one articulation point. At least one of these leaf blocks has no other special vertices and can be reduced by Rule 1. ◀

► **Lemma 12.** *Let  $f$  be the number of edges between  $S$  and those connected components of  $G - S$  which have at least 3 edges to  $S$ . If  $f \geq 24k$  then  $(G, k)$  is a YES-instance.*

**Proof.** Let  $U_1, \dots, U_m$  be a connected components of  $G - S$ . Without loss of generality we can assume that the first  $l$  of them have more than three edges to  $S$  and the other have only 2. Note that  $l \leq \frac{f}{3}$ .

If  $f \geq 24k$  we construct a cut of  $G$  with at least  $|V(G)| + k$  edges. First of all, in each connected component  $U_i$  we construct a cut of size at least  $|V(U_i)| - 1$  according to a spanning tree lower bound. After that we put the whole  $S$  into one of the parts so that at least half of edges between  $S$  and  $G - S$  is cut. Note that number of edges between  $S$  and  $G - S$  is at least  $2(m - l) + f$ .

So, the constructed cut has the following size:

$$\begin{aligned} \text{mc}(G) &\geq \sum_{i=1}^m (|V(U_i)| - 1) + (m - l) + \frac{f}{2} = \sum_{i=1}^m |V(U_i)| - l + \frac{f}{2} \geq \sum_{i=1}^m |V(U_i)| - \frac{f}{3} + \frac{f}{2} \\ &= \sum_{i=1}^m |V(U_i)| + \frac{f}{6} \geq \sum_{i=1}^m |V(U_i)| + 4k \geq \sum_{i=1}^m |V(U_i)| + |S| + k \geq V(G) + k \end{aligned}$$

Constructed cut has size at least  $\geq V(G) - 1 + k$  which means that  $(G, k)$  is a Yes-instance. ◀

Before we state the next reduction rule we prove the following auxiliary claim.

▷ **Claim 13.** Let  $M$  be a maximum cut in a graph  $G$ , and edge  $e$  is a bridge in  $G$  then  $e \in M$ .

Proof. If  $e \in M$  then there is nothing to prove. Otherwise,  $e \notin M$ . Let  $e = uv$ . The maximum cut  $M$  induces a partition of vertices in  $G$  into two parts  $V_1, V_2$ . Without loss of generality  $u, v \in V_1$ . The edge  $e$  also induces a partition of  $V(G)$  into two parts  $W_1, W_2$ , each part corresponds to a connected component in a graph  $G - e$ . Without loss of generality, we assume that  $u \in W_1, v \in W_2$ . Let us consider the following partitioning  $(A, B)$  where  $A = (V_1 \cap W_1) \cup (V_2 \cap W_2), B = (V_1 \cap W_2) \cup (V_2 \cap W_1)$ . Note that  $u \in A, v \in B$ . So the cut generated by partitioning  $(A, B)$  contains edge  $e$ . Moreover, if edge  $e' \neq e$  then either  $e'$  connects two vertices from  $W_1$  or two vertices from  $W_2$ . Hence, if  $e' \in M$  then  $e'$  has one endpoint in  $A$  and the other in  $B$ . So,  $M$  is not a maximum cut which leads to a contradiction. ◁

<b>Rule 3</b>	Let $U$ be a block in $G - S$ which is a clique, $W \subseteq U$ is the set of special vertices such that $ W  \leq \lceil  U /2 \rceil$
Remove:	Vertices $V(U) \setminus W$ and a maximal subset of edges $F$ in $G[W]$ such that $(G - (U - W)) \setminus F$ is connected
Parameter:	Reduce $k$ by $\text{mc}(U) -  V(U)  +  W  -  E(G[W] \setminus F) $

**Proof of correctness.** Let  $G'$  be a graph obtained from  $G$  by deletion of all edges from  $U$  and vertices  $V(U) \setminus W$ .

Since  $U$  is a clique, its maximum cut is achieved when its vertices are divided between partitions equally (or almost equally if  $|U|$  is odd). Since  $|W| \leq \lceil |U|/2 \rceil$ , any cut of  $G'$  can be extended to induced maximum cut of  $U$  (simply add vertices from  $U \setminus W$  to both parts such that they contain  $\lceil |U|/2 \rceil$  and  $\lfloor |U|/2 \rfloor$  vertices from  $U$ ). Therefore,  $\text{mc}(G') + \text{mc}(U) \leq \text{mc}(G)$ .

Since  $G'$  and  $U$  do not have common edges we have  $\text{mc}(G') + \text{mc}(U) \geq \text{mc}(G)$ . Combining the two inequalities we have  $\text{mc}(G') + \text{mc}(U) = \text{mc}(G)$ .

Let  $G''$  be a graph  $(G - (U - W)) \setminus F$ . Note that  $V(G'') = V(G')$  and  $E(G'') = E(G') \cup (E(G[W]) \setminus F)$ . Each edge from the set  $E(G[W]) \setminus F$  is a bridge in  $G''$ , otherwise  $F$  is not maximal. In any graph maximum cut contains all bridges, by Claim 13, so we have:  $\text{mc}(G'') = \text{mc}(G') + |E(G[W]) \setminus F| = \text{mc}(G) - \text{mc}(U) + |E(G[W]) \setminus F|$ . And this proves correctness of the reduction.

Note that the reduction preserves the clique-cycle-forest property of  $G - S$  as we delete the subset  $V(U) \setminus W$  and the clique  $W$  was replaced by a spanning tree. So each edge in the spanning tree of  $W$  can be seen as a clique on two vertices. ◀

<b>Rule 4</b>	Let $e = bc$ be an edge from $E(G - S)$ such that $\deg_G b = \deg_G c = 2$ , and $a, d$ be the other neighbours of $b$ and $c$ respectively (note that $a$ and $d$ may be the same vertex)
Remove:	vertices $b, c$ and add edge $ad$ if $a \neq d$
Parameter:	No changes applied

**Proof of correctness.** Let  $G'$  be graph obtained after application of the reduction rule. We prove that  $\text{mc}(G') = \text{mc}(G) - 2$ .

First, we show that  $\text{mc}(G') \geq \text{mc}(G) - 2$ . Let  $(A, B)$  be partition of  $G$  that produces the maximum cut. Induced partition in the subgraph  $G'$  has cut of size  $\geq \text{mc}(G) - 2$ . Indeed, if all 3 edges  $ab, bc, cd$  are in cut  $E(A, B)$ , then  $a, d$  are in different partitions. Hence, the

induced partition cuts an edge  $ad$  in  $G'$  having a cut of size  $\text{mc}(G) - 3 + 1 = \text{mc}(G) - 2$  in  $G'$ . If at most 2 edges from the set  $\{ab, bc, cd\}$  belong to the cut  $E(A, B)$  then obviously induced vertex partition has cut at least  $|E(A, B)| - 2 = \text{mc}(G) - 2$ .

To prove that  $\text{mc}(G) \geq \text{mc}(G') + 2$  we construct partitioning of vertices  $V(G)$  based on partitioning of vertices  $V(G')$ . Let  $(A', B')$  be the maximum cut in  $G'$ . We construct partition of  $V(G)$  simply by adding vertices  $b, c$  to partition  $(A', B')$ . We place  $c$  be in the same part as  $a$  and  $b$  be in the other. Let us call obtained partition  $(A, B)$ . In this case edges  $ab, bc$  belong to cut  $E(A, B)$ , edge  $cd$  is in the cut if and only if  $ad$  was in the cut of the graph  $G'$ . Hence,  $|E_G(A, B)| = |E_{G'}(A', B')| + 2$ .

Since the lower bound and the maximum cut size decrease by two, the parameter should not be changed.

Note that after application of the reduction rule  $G' - S$  is a clique-cycle-forest as  $G - S$ . However, the reduction can remove a cycle of length 3 or turn a cycle of size 4 into two multiple edges. ◀

► **Lemma 14.** *If none of the above rules is applicable to  $G$  then at least half of all vertices in every block of  $G - S$  are special.*

**Proof.** For cliques it is true since otherwise rule 3 is applicable. Cycles can not have 2 consecutive non-special vertices since otherwise we can apply reduction rule 4. ◀

**Rule 5** If there are at least  $k$  even cycles blocks in  $G - S$   
 Return: YES-instance.

**Proof of correctness.** Consider a subgraph  $H$  containing  $k$  even cycles of  $G - S$ . We construct a spanning forest in  $H$ , to do this we remove an arbitrary edge in each cycle. This forest can be extended to a spanning tree  $F$  of the whole  $G$ . If we add back previously deleted edges of the subgraph  $H$  we obtain a bipartite graph (all introduced cycle have even length). The bipartite subgraph has exactly  $|V(G)| - 1 + k$  edges, hence,  $G$  has a required cut and we can output YES. ◀

Now we can bound the total size of connected components in  $G - S$  that have at least 3 edges to  $S$  (note that here we are speaking about connected components and not blocks). We split blocks of these components into several types and bound the total number of vertices in block of each type. We consider the following types of blocks from connected components with at least 3 edges going to  $S$ :

1. Blocks with  $\geq 3$  articulation points;
2. Blocks with  $< 3$  articulation points and at least one special vertex that is not an articulate vertex;
3. Even cycles with exactly two special vertices which are articulation points;
4. Cliques and odd cycles with exactly two special vertices which are articulation points.

These types of blocks cover all possibilities, since in connected components with at least three edges towards  $S$  all blocks with only one special vertex were removed by rule 1. Below for each type we bound the total size of blocks of this type.

1. For the graph  $G - S$  we consider a bipartite forest of blocks and articulation points  $F$ : for every block  $X$  we introduce a vertex  $a_X$  and for every articulation point  $y$  we introduce a vertex  $b_y$ . Moreover,  $a_X b_y \in E(F)$  if and only if  $y \in V(X)$ . Each articulation point belong to at least two blocks. Therefore leaves and isolated vertices in  $F$  correspond to blocks with one and zero articulation points respectively. We recall, that we removed all

blocks with at most one special vertex except isolated vertices with at least two edges going to  $S$ . Hence, each block corresponding to isolated vertex or leaf in the forest  $F$  has an edge going to  $S$ , as every non-isolated block left has at least two special vertices each of these blocks has its own edge to  $S$ . Hence, by Lemma 12 we have that there are at most  $24k$  leaves and isolated vertices in  $F$ .

It is known that for any tree or forest  $T$  the following holds

$$\sum_{v \in T: \deg_T v > 2} (\deg(v) - 2) \leq \text{number of leaves in } T.$$

Hence, we have

$$\sum_{v \in F: \deg_F v > 2} (\deg_F(v) - 2) \leq 24k. \quad (1)$$

Moreover,  $\sum_{v \in F: \deg_F v > 2} \deg_F(v) \leq 3 \sum_{v \in F: \deg_F v > 2} (\deg_F(v) - 2) \leq 3 \cdot 24k = 72k$ .

Note that the total number of articulation points in blocks with at least 3 articulation points is bounded by  $\sum_{v \in F: \deg_F v > 2} \deg_F(v)$ . The total number of special vertices that are not articulation points in such blocks is bounded by  $24k$ , by Lemma 12. Therefore, the total number of special vertices in blocks with at least 3 articulation points is bounded by  $72k + 24k = 96k$ . Hence, by Lemma 14 these blocks have at most  $2 \cdot 96k = 192k$  vertices.

2. Each block of these type has at least one special vertex  $v$  that is not an articulation point. This vertex must have an edge to  $S$ . Therefore at least one third of special vertices in this type blocks are vertices with an edge to  $S$ . By Lemma 12, number of vertices with an edge to  $S$  is at most  $24k$ . Hence, blocks of this type have at most  $3 \cdot 24k = 72k$  special vertices. Moreover, by lemma 14 we conclude that the total number of vertices in these blocks is at most  $2 \cdot 72k = 144k$ .
3. There is at most  $k$  such blocks otherwise the reduction rule 5 is applicable. By Lemma 14 each of these blocks has at most 4 vertices, so their total size is at most  $4k$ .
4. We note, that the only blocks in this category are simple edge between two articulation points. Indeed, a clique having more vertices would be reduced by Rule 3. An odd cycle with only two special vertices must have size 3 by Lemma 14. However, a cycle on 3 vertices is also a clique, so it is also subject to reduction by rule 3.

To bound the number of vertices in these blocks we again use forest of blocks and articulation points  $F$  constructed before. Let us mark and count vertices in this forest that belong to one of the following classes:

- a. **vertices of degree  $\geq 3$ :** there is at most  $24k$  of such vertices, follows from inequality 1
- b. **other vertices corresponding to blocks of type 2 and 3:** Note that blocks of first type have degree  $\geq 3$  and therefore are counted in previous point. Before we showed that there are at most  $24k$  blocks of the second type and at most  $k$  blocks of the third type, so there are at most  $25k$  vertices of these type.
- c. **vertices corresponding to articulation points with an edge to  $S$ :** by Lemma 12 there is at most  $24k$  such vertices.

Recall that we mark all vertices from bullets  $a, b, c$ . Hence, in total there are at most  $24k + 25k + 24k = 73k$  marked vertices. Note that every block of type 4 corresponds to an unmarked vertex.

Unmarked vertices correspond to blocks of type 4 and articulation points that (i) does not have an edge to  $S$ , (ii) belongs to exactly two blocks. We cannot have a path of unmarked vertices of length 6 since otherwise Rule 4 is applicable.

Note that every unmarked vertex is of degree 2. We consider a forest  $F'$  obtained from  $F$  by contracting every path of unmarked vertices into an edge.  $F'$  is a forest with only marked vertices so it has at most  $73k$  vertices and at most  $\leq 73k - 1$  edges. No more than 6 unmarked vertices can be mapped to a single edge of  $F$ . Hence, there are at most  $6(73k - 1) = 438k - 6$  unmarked vertices in  $F$ . Recall that number of blocks of type 4 is at most the number of unmarked vertices. Moreover, each such block contains at most 2 vertices. Hence, number of vertices in such blocks is at most  $876k - 12$ .

We just have shown that number of vertices in connected components with at least three edges to  $S$  is at most  $O(k)$ . Now it is left to obtain a bound for the total size of connected components that have exactly 2 edges to  $S$  (one edge is not possible due to Rules 1, 2, zero edges is impossible due to connectivity of the graph). From now on we are considering only blocks of connected components that have exactly 2 edges to  $S$ . Since each leaf block has an edge to  $S$ , these components are either a single block with 2 edges to  $S$ , or a chain of blocks with two leaf blocks having exactly one edge to  $S$ , or a single vertex having two edges going to  $S$ .

Note that every block in these components has exactly 2 special vertices or the whole component is a single vertex with two edges going to  $S$ . Hence, each such block is a cycle  $C_4$ , or a cycle  $C_2$  (a double edge), or a single edge  $K_2$ , or the whole component is just one vertex.

**Rule 6** If the total number of vertices in the connected components of  $G - S$  with 2 edges to  $S$  is at least  $20k$   
 Return: YES.

**Proof of correctness.** Let  $S' = S \cup$  (even cycle blocks of connected components of  $G - S$  with 2 edges to  $S$ ) and  $U =$  other vertices of these components. Due to Rule 5 we have at most  $k$  even cycles in  $G - S$ . Moreover, we are considering blocks that have at most 4 vertices. Hence, at most  $4k$  vertices can belong to an even cycles. Therefore,  $|S'| \leq 3k + 4k = 7k$  and  $|U| \geq 20k - 4k \geq 16k$ .

Every vertex in  $U$  has a degree 2 in  $G$ . Hence, there are no edges between vertices in  $U$ , since otherwise to this edge Rule 4 is applicable. Therefore each vertex in  $U$  has 2 edges going to  $S'$ . We prove that in this case the graph contains a cut of size at least  $V(G) + k$ . To do this we consider the following procedure:

- take a spanning forest  $H$  in  $G$  such that each tree in the forest has exactly one vertex in  $S'$ ;
- uniformly randomly partition vertices in  $S'$  in two parts;
- place the rest of all vertices in the part so that all edges from trees in  $H$  are in cut.

Note that every two vertices from different trees are placed in parts randomly independently. Hence, each edge between trees can be in the constructed cut with probability  $\frac{1}{2}$ . Since  $|S'| \leq 7k$  and each tree has one vertex in  $S'$  there is at least  $|V(G)| - 7k$  edges in  $H$ . Every vertex in  $U$  has two edges to vertices in  $S'$ . Hence, one of the edges is an edge between two trees in  $H$ . Therefore, the expected number of edges in the cut is at least  $\geq |E(H)| + \frac{16k}{2} = |V(G)| - |S'| + \frac{16k}{2} \geq |V(G)| - 7k + \frac{16k}{2} = |V(G)| + k$ . Hence, we proved that a cut of size at least  $|V(G)| - 1 + k$  exists.

Note that this procedure can be derandomized using method of conditional probabilities [17] which allows to find a required cut deterministically, if needed. ◀

Now we are ready to bound the overall number of vertices. Recall that there are at most  $192k$  vertices in blocks of type 1, at most  $144k$  vertices in blocks of type 2, at most  $4k$  vertices in blocks of type 3, at most  $876k - 12$  vertices in blocks of type 4. Hence,

number of vertices in connected components with at least 3 edges going to  $S$  is at most  $192k + 144k + 4k + 876k - 12 = 1216k - 12$ . We also have at most  $20k$  vertices in other connected components of  $G - S$  and at most  $3k$  vertices in  $S$ . In total we have at most  $1216k - 12 + 20k + 3k = \mathcal{O}(k)$  vertices.

In order to bound number of edges we employ the final reduction rule.

**Rule 7** If graph contains at least  $2k + 2|V(G)|$  edges  
Return: YES-instance.

**Proof of correctness.** Every graph has a cut of size at least  $\frac{|E(G)|}{2}$ . Hence, in this case graph has a cut of size at least  $\frac{2k+2|V(G)|}{2} = |V(G)| + k$  which means that our input is a YES-instance. ◀

Taking into account that  $|V(G)| = \mathcal{O}(k)$  after application of Rule 7 we conclude that  $|E(G)| = \mathcal{O}(k)$ .

The algorithm either produces a graph with  $\mathcal{O}(k)$  vertices and edges or concludes that the graph has a required cut. In the latter we can output a trivial YES-instance. Hence, we constructed a linear kernel.

Note that the final graph might contain multi-edges. However, in this case we can apply reduction rule 4 backwards and replace every multi-edge with a cycle of length 4. It is easy to see that exhaustive application of this operations increase kernel size only linearly. ◀

The following theorem shows that the obtained kernel has asymptotically optimal size unless ETH fails.

► **Theorem 2.** *There is no kernel with sublinear number of vertices for MAX-CUT-AST unless ETH fails.*

**Proof.** A trivial brute force algorithm solves MAX-CUT in  $\mathcal{O}(2^{|V(G)|})$ . A kernel with  $o(k)$  vertices would imply that we can solve MAX-CUT in  $2^{o(k)} = 2^{o(|E(G)|)}$  which is impossible unless ETH fails. ◀

## 4 Subexponential Algorithms for Max-Cut-AST

In this section we present subexponential algorithms for MAX-CUT-AST in chordal and co-bipartite input graphs.

### 4.1 Co-bipartite Graphs

► **Theorem 3.** *MAX-CUT-AST can be solved in a connected co-bipartite graph  $G$  in  $2^{\mathcal{O}(\sqrt{k})}$  poly time.*

**Proof.** Note that co-bipartite graphs consist of two disjoint cliques and edges between them. At least one of the cliques contains at least half of the vertices. Therefore, each co-bipartite graph contains at least  $\frac{n}{2}(\frac{n}{2} - 1)/2$  edges. It is well known that a maximum edge cut in any graph contains at least half of all edges. Hence, in any co-bipartite graph maximum cut size is at least  $\frac{n}{2}(\frac{n}{2} - 1)/4$ . Therefore, if  $\frac{n}{2}(\frac{n}{2} - 1)/4 \geq n - 1 + k$  we can output Yes immediately. Otherwise,  $k = \Omega(n^2)$  and the simple brute-force algorithm with working time  $2^n n^{\mathcal{O}(1)}$  is actually  $2^{\mathcal{O}(\sqrt{k})}$  poly algorithm. ◀

## 4.2 Chordal Graphs

First of all we recall some well-known results and prove auxiliary lemma.

► **Lemma 17** ([11]). *For an input chordal graph  $G$  we can find a tree decomposition of minimum width in polynomial time. Moreover, in this tree decomposition all vertices of each bag induce a clique in the original input graph  $G$ .*

► **Lemma 18** ([1]). *Given an input graph  $G$  and its tree decomposition of width  $k$  one can find the size of maximum cut in  $2^{\mathcal{O}(k)}$  poly time.*

► **Lemma 19.** *Let  $H$  be an induced subgraph in a connected graph  $G$  and graph  $H$  has a cut with at least  $\ell$  edges then the graph  $G$  has a cut of size at least  $|V(G)| - |V(H)| + \ell$ .*

**Proof.** Assume that  $(A, B)$  is the partitioning of  $V(H)$  into two parts such that  $|E(A, B)| \geq \ell$ . We complete the partitioning  $(A, B)$  with vertices from  $G \setminus H$  such that we get a cut of size at least  $|V(G)| - |V(H)| + \ell$ .

Let us consider a spanning forest  $F$  of the graph  $G$  such that each tree  $T$  from the forest  $F$  has only one vertex from  $V(H)$  and the vertex is a root node of the tree. For each vertex  $v \in V(G)$  we assign a tree  $T_v$  such that  $v \in T_v$  and  $T_v \in F$ . Moreover, for each  $v \in V(G)$  we assign vertex  $r_v \in V(H)$  such that  $r_v \in T_v$  and  $r_v$  is a root vertex of  $T_v$ . Now we are ready to present the partitioning. If  $d_{T_v}(v, r_v)$  is even we place  $v$  in the same part as  $r_v$ , otherwise we place  $v$  in the opposite part of  $r_v$ . In the constructed partitioning each edge of forest  $F$  is in the cut, since for each vertex its ancestor belong to a different part. Since the graph  $G$  is connected we have that  $|E(F)| = |V(G)| - |V(H)|$ . Also the partitioning contains all edges from the initial cut of subgraph  $H$ . Therefore, we constructed a cut of size at least  $|V(G)| - |V(H)| + \ell$ . ◀

Now we have all ingredients to prove the main theorem of this subsection.

► **Theorem 4.** *MAX-CUT-AST on a connected chordal graphs admits a  $2^{\mathcal{O}(\sqrt{k})}$  poly-time algorithm.*

**Proof.** Using Lemma 17 we find tree decomposition of the input graph  $G$ . In this tree decomposition each bag is a clique. We denote by  $\ell - 1$  the treewidth of the input graph  $G$ . Hence, there is a bag of size  $\ell$ . It means that  $G$  contains a subgraph  $H$  such that  $H$  is a clique on  $\ell$  vertices. If  $\frac{\ell(\ell-1)}{4} \geq k + \ell - 1$  then by Lemma 19 we conclude that the graph  $G$  contains a cut of size at least  $n + k - 1$ .

Otherwise,  $\frac{\ell(\ell-1)}{4} < k + \ell - 1$  and we have  $\ell < 5 + 2\sqrt{k}$ . Hence,  $\ell = \mathcal{O}(\sqrt{k})$  and we can find an exact value of the maximum cut by standard dynamic programming using tree decomposition. The running time of this algorithm is  $2^{\mathcal{O}(\ell)}$  poly =  $2^{\mathcal{O}(\sqrt{k})}$  poly. ◀

► **Corollary 21.** *MAX-CUT-AST on a connected split graph admits a  $2^{\mathcal{O}(\sqrt{k})}$  poly algorithm.*

**Proof.** The statement immediately follows from the previous theorem since the class of all split graphs is a subclass of all chordal graphs. ◀

## 4.3 Lower Bounds

Under assumption of the Exponential Time Hypothesis we show that algorithms for chordal and split graphs are essentially tight.

► **Theorem 5.** *MAX-CUT-AST does not admit  $2^{\mathcal{O}(\sqrt{k})}$  poly or  $2^{\mathcal{O}(\sqrt{n})}$  poly algorithms on split graphs and on chordal graphs unless the ETH is false.*

**Proof.** Bodlaender and Jansen [1] presented a reduction of MAX-CUT on arbitrary graph  $G$  to MAX-CUT on split graph. This reduction transform graph  $G$  with a cut  $K$  into a graph  $G'$  on  $|V(G)| + |E(\bar{G})|$  vertices with  $|V(G)| \cdot (|V(G)| - 1)/2 + 2|E(\bar{G})|$  edges. Moreover,  $G$  has the maximum edge cut  $K$  if and only if  $G'$  has the maximum edge cut  $2|E(\bar{G})| + K$ . Note that  $2|E(\bar{G})| + K = O(n^2)$  and  $|V(G)| + |E(\bar{G})| = O(n^2)$ . Hence, by solving MAX-CUT-AST in  $2^{o(\sqrt{k})}$  poly time (or in  $2^{o(\sqrt{|V(G')|})}$  poly) on split graphs we can solve MAX-CUT on arbitrary graphs in  $2^{o(n)}$  poly time. However, it is known that under assumption of ETH MAX-CUT does not admit  $2^{o(n)}$  poly algorithm [6]. So we get a desired contradiction. ◀

► **Theorem 6.** *MAX-CUT-AST does not admit  $2^{o(\sqrt[4]{k})}$  poly algorithm on co-bipartite graphs unless the ETH is false.*

**Proof.** Bodlaender and Jansen [1] presented a reduction of MAX-CUT on split graphs to MAX-CUT on co-bipartite graph. If the initial split graph  $G$  contains  $n$  vertices then the obtained co-bipartite graph will contain  $O(n)$  vertices and  $O(n^2)$  edges. It means that  $k = O(n^2)$ . If MAX-CUT-AST admits  $2^{o(\sqrt[4]{k})}$  poly =  $2^{o(\sqrt[4]{n^2})}$  poly =  $2^{o(\sqrt{n})}$  poly algorithm on co-bipartite graphs then we can solve MAX-CUT on split graphs in  $2^{o(\sqrt[4]{k})}$  poly =  $2^{o(\sqrt[4]{n^2})}$  poly =  $2^{o(\sqrt{n})}$  poly which contradict the previous theorem. ◀

## 5 Tripartite Graphs

### 5.1 Parametrization Above a Spanning Tree

► **Lemma 24.** *Let us assume that MAX-CUT-AST on tripartite graphs with  $n$  vertices admits algorithm with running time  $T(k, n)$  where  $k$  is the parameter. Then MAX-CUT-AST on arbitrary graph  $G$  with  $n'$  vertices and  $m'$  edges can be solved in  $T(k, n' + 2m') + \text{poly}(n')$  time.*

**Proof.** To prove the statement we reduce MAX-CUT-AST on arbitrary graph  $G$  to MAX-CUT-AST on tripartite graph  $G'$  such that  $|V(G')| = |V(G)| + |2E(G)|$ . We construct  $G'$  in the following way, for each edge  $e = uv \in E(G)$ : (i) create two vertices  $e_u, e_v$ ; (ii) delete edge  $uv$ ; (iii) add edges  $ue_u, e_u e_v, e_v v$ . Essentially we subdivide each edge twice. It is easy to see that  $V(G)$  is an independent set in graph  $G'$  and  $G' \setminus G$  is a disjoint union of edges. Hence,  $G'$  is a tripartite graph.

Now we provide a connection between the size of maximum cut in the graph  $G$  and the size of maximum cut in  $G'$ . We claim that  $\text{mc}(G') = \text{mc}(G) + 2|E(G)|$ . Having partition  $(A', B')$  of vertices  $V(G')$  we construct partitioning  $(A, B)$  of  $V(G)$  such that  $|E(A, B)| \geq |E(A', B')| - 2|E(G)|$ . We take  $A = A' \cap V(G), B = B' \cap V(G)$ . All edges in  $E(G')$  we can partition in triples of the following type  $ue_u, e_u e_v, e_v v$ . It is easy to see that if cut  $E(A', B')$  contains all three edges  $ue_u, e_u e_v, e_v v$  then cut  $E(A, B)$  contains at least one edge. So it means that for each triple we lose at most 2 edges in  $E(A, B)$  compared to  $E(A', B')$ . Hence,  $|E(A, B)| \geq |E(A', B')| - 2|E(G)|$ .

Now, it is enough to prove that having a cut  $(A, B)$  of graph  $G$  we can construct a cut  $(A', B')$  of  $G'$  such that  $|E(A', B')| \geq |E(A, B)| + 2|E(G)|$ . In order to do this for each edge  $e = uv$  we place vertices  $e_u, e_v$  in the opposite part to vertices  $u, v$  correspondingly, i.e. if  $u \in A, v \in B$  then we put  $u, e_v$  to  $A'$  and  $v, e_u$  to  $B'$ , similarly, if  $u, v \in A$  then we put  $u, v$  to  $A'$  and  $e_u, e_v$  to  $B'$ . It is easy to see that for such cut  $(A', B')$  we have  $|E(A', B')| \geq |E(A, B)| + 2|E(G)|$ .

So we proved that  $\text{mc}(G') = \text{mc}(G) + 2|E(G)|$ . Recall that  $|V(G')| = |V(G)| + |2E(G)|$ . Hence,  $\text{mc}(G') - (|V(G')| - 1) = \text{mc}(G) - (|V(G)| - 1)$ . Therefore, answer for MAX-CUT-AST on graph  $G'$  is the same as the answer for MAX-CUT-AST on graph  $G$ . As  $G'$  is tripartite we have proved the desired result. ◀

Above lemma implies the following result.

► **Corollary 25.** *There is no  $2^{o(k)}$  poly algorithm for MAX-CUT-AST on tripartite graphs under assumption of the ETH.*

**Proof.** Such algorithm would imply that MAX-CUT-AST can be solved on arbitrary graph in  $2^{o(k)}$  poly time by lemma 24. However, existence of such algorithm contradict to the ETH [15]. ◀

## 5.2 Parameterization Above $\frac{2}{3}|E|$

First of all we show that any tripartite graph  $G$  has a maximum cut of size at least  $\frac{2}{3}|E|$ . Let assume that  $G$  has parts  $V_1, V_2, V_3$ . It is easy to see that  $|E(V_1, V_2 \cup V_3)| + |E(V_2, V_1 \cup V_3)| + |E(V_3, V_1 \cup V_2)| = 2|E(G)|$ . So at least one of the cuts  $(V_1, V_2 \cup V_3), (V_2, V_1 \cup V_3), (V_3, V_1 \cup V_2)$  has size  $\frac{2}{3}|E|$ . However, even knowing that a graph is tripartite we may not have a partition itself.

Below we show that we can find an edge cut of size  $\frac{2}{3}|E|$  in tripartite graphs even if partition on three parts is not provided in the input. To get this result we employ semi-definite programming.

► **Theorem 7.** *There is a polynomial time randomized algorithm that finds an edge cut of size  $\frac{2}{3}|E(G)|$  on an input tripartite graph  $G$ .*

**Proof.** First of all for each vertex  $v_i \in G$  we assign vector  $\vec{v}_i \in \mathbb{R}^n$  and formulate the following program:

$$\begin{aligned} & \min \lambda \\ & \forall i \in V(G) \langle \vec{v}_i, \vec{v}_i \rangle = 1 \\ & \forall ij \in E(G) \langle \vec{v}_i, \vec{v}_j \rangle \leq \lambda \end{aligned}$$

Essentially we are looking for  $n$  unit vectors such that minimum angle between any two vectors that correspond to adjacent vertices is maximized. We know that the input graph is tripartite, hence there is a solution such that  $\lambda \leq -\frac{1}{2}$ . To achieve the value  $-\frac{1}{2}$  it is enough to map vertices of first part into  $(1, 0, \dots, 0)$ , vertices of second part into  $(-\frac{1}{2}, \frac{\sqrt{3}}{2}, 0, \dots, 0)$ , vertices of third part into  $(-\frac{1}{2}, -\frac{\sqrt{3}}{2}, 0, \dots, 0)$ . It means that in the optimum solution of the semidefinite program angle between two vectors corresponding to two adjacent vertices is at least  $\frac{2\pi}{3}$ . It is known [18] that for any  $\epsilon$  semidefinite programming can be solved with additive error of size  $\epsilon$  in polynomial time from the input and  $\log \frac{1}{\epsilon}$ . Therefore, we can find the optimum value of  $\lambda$  up to additive error  $\epsilon$ . After that we generate a random hyperplane and split vertices of graph into two parts. We put vertex  $v_i$  to the first part if the vector  $\vec{v}_i$  is lying on one side of the generated hyperplane or to the second if  $\vec{v}_i$  is lying on the other side of the hyperplane. The expected value of an edge  $e = v_i v_j$  being in the cut under such partitioning is equal to  $\arccos(\langle \vec{v}_i, \vec{v}_j \rangle) \geq \arccos(-\frac{1}{2} + \epsilon) = \frac{2}{3} - \delta(\epsilon)$  [18, Chapter 6]. Hence, the expected value of the cut is at least  $(\frac{2}{3} - \delta(\epsilon))|E|$ . We choose  $\epsilon$  in such way that  $\delta(\epsilon)|E| \leq \frac{1}{6}$ . Since derivative of  $\arccos(\cdot)$  is bounded in the neighborhood of  $-\frac{1}{2}$  we can take  $\epsilon = \frac{1}{1000|E|^2}$ . If we choose  $\epsilon$  in this way we obtain a polynomial time algorithm which output a cut of expected value at least  $\frac{2}{3}|E| - \frac{1}{6}$ . Note that the value  $\frac{2}{3}|E| - \frac{1}{6}$  is not integer and the size of a cut is an integer. So, the algorithm sometimes must output values of size at least  $\lceil \frac{2}{3}|E| \rceil - \frac{1}{6} = \lceil \frac{2}{3}|E| \rceil$ . Since the cut size is bounded by  $|E|$ , our algorithm output value of size  $\lceil \frac{2}{3}|E| \rceil$  with probability at least  $\Omega(\frac{1}{|E|})$ . Therefore it is enough to repeat the algorithm polynomial number of times, take the largest cut among all generated. The algorithm will not output cut of size at least  $\lceil \frac{2}{3}|E| \rceil$  only with exponentially small probability. ◀

► **Lemma 27** (\*). *Under the assumption  $P \neq NP$  there is no polynomial time algorithm that for each tripartite graph  $G$  finds a cut of size at least  $\frac{5}{6}|E|$  if it exists.*

In the following theorem we show that fraction  $\frac{5}{6}$  can be replaced with any fraction of the following type  $\frac{2}{3} + \epsilon$  where  $\epsilon \in (0, \frac{1}{6})$ . Informally speaking to achieve this results it is enough to add large number of disjoint subgraphs  $K_3$  to the construction in the previous theorem. Indeed, each partition of a triangle cuts at most 2 edges out of 3. Therefore, by adding triangles we make fraction of size of maximum cut to the total number of edges being close to  $\frac{2}{3}$ .

► **Theorem 8.** *Assuming  $P \neq NP$  for any  $0 < \epsilon < \frac{1}{6}$  there is no polynomial time algorithm that finds cut of size  $\geq (\frac{2}{3} + \epsilon)|E|$  in tripartite graph if such cut exists.*

**Proof.** In order to refute existence of such algorithm we show that using this algorithm we can construct a polynomial time algorithm that on tripartite graphs with the number of edges divisible by 6 outputs cut of size at least  $\frac{5}{6}|E|$ , if such cut exists.

Let us assume that tripartite graph  $G$  contains  $n$  vertices and  $m$  edges and we need to check existence of cut of size at least  $\frac{5}{6}m$ . If we add  $k$  disjoint triangles to the graph  $G$  then essentially we need to check if in the new graph exists a cut of size at least  $\geq \frac{5m}{6} + 2k$ , since in each triangle we can cut at most two edges. However, the number of edges in the new graph is  $m + 3k$ . Therefore the ratio between the cut and the overall number of edges becomes  $\frac{5m/6+2k}{m+3k}$ . This fraction tends to  $\frac{2}{3}$  as long as we increase  $k$ . Consider the maximum integer  $k$  such that the fraction is not smaller than  $\frac{2}{3} + \epsilon$ . Note that, for fixed  $\epsilon$  we have  $k = \mathcal{O}(m)$ . Since we chose the maximum  $k$  with such properties we have that:

$$\frac{5m/6 + 2(k+1)}{m + 3(k+1)} < \frac{2}{3} + \epsilon \quad (2)$$

Now it is left to show that cuts with  $\frac{5}{6}m - 1$  edges does not satisfy required equation, i.e. that:

$$\frac{5m/6 + 2k - 1}{m + 3k} < \frac{2}{3} + \epsilon$$

To do this it is enough to show that  $\frac{5m/6+2k-1}{m+3k} < \frac{5m/6+2(k+1)}{m+3(k+1)}$ . Below we provide detailed computations.

$$\begin{aligned} & \frac{5m/6 + 2k - 1}{m + 3k} < \frac{5m/6 + 2(k+1)}{m + 3(k+1)} \\ 0 & < \frac{5m/6 + 2k + 2}{m + 3k + 3} - \frac{5m/6 + 2k - 1}{m + 3k} \\ 0 & < \frac{(5m/6 + 2k + 2)(m + 3k) - (5m/6 + 2k - 1)(m + 3k + 3)}{(m + 3k + 3)(m + 3k)} \\ 0 & < \frac{(5m/6 + 2k)(m + 3k) + 2(m + 3k) - (5m/6 + 2k)(m + 3k + 3) + (m + 3k + 3)}{(m + 3k + 3)(m + 3k)} \\ 0 & < \frac{(5m/6 + 2k)(m + 3k) + 2(m + 3k) - (5m/6 + 2k)(m + 3k) - 3(5m/6 + 2k) + (m + 3k + 3)}{(m + 3k + 3)(m + 3k)} \\ 0 & < \frac{2(m + 3k) - 3(5m/6 + 2k) + (m + 3k + 3)}{(m + 3k + 3)(m + 3k)} \\ 0 & < \frac{m(2 - 5/2 + 1) + k(6 - 6 + 3) + 3}{(m + 3k + 3)(m + 3k)} \\ 0 & < \frac{m/2 + 3k + 3}{(m + 3k + 3)(m + 3k)} \end{aligned}$$

Hence, the constructed graph with disjoint triangles  $G'$  contains cut of size  $\frac{5m}{6} + 2k$  if and only if it contains at least  $(\frac{2}{3} + \epsilon)|E(G')|$ . Moreover, it happens if and only if initial graph contained a cut with at least  $\frac{5}{6}|E(G)|$  edges. ◀

## 6 Conclusion

In Section 5 we initiate study of MAXIMUM CUT problem on tripartite graphs parameterized above  $\frac{2}{3}|E|$  lower bound. We think that existence of an FPT algorithm for this problem is an interesting open question.

---

### References

- 1 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. In Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner, editors, *STACS 94*, pages 769–780, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- 2 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.
- 3 Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *Theoretical Computer Science*, 513:53–64, 2013.
- 4 Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the edwards-erdős bound. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I 39*, pages 242–253. Springer, 2012.
- 5 Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the edwards-erdős bound. *Algorithmica*, 72(3):734–757, 2015.
- 6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. URL: <https://www.springer.com/gp/book/9783319212746>.
- 7 R. Diestel. *Graph Theory*. Springer, 2006.
- 8 Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. *Algorithmica*, 80:2574–2615, 2018.
- 9 Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. *Algorithmica*, 80:2574–2615, 2018.
- 10 MR Garey, DS Johnson, and L Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.
- 11 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. doi:10.1016/0095-8956(74)90094-X.
- 12 Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.
- 13 Gregory Gutin and Matthias Mnich. A survey on graph problems parameterized above and below guaranteed values. *arXiv preprint arXiv:2207.12278*, 2022.
- 14 Richard M Karp et al. Complexity of computer computations. *Reducibility among combinatorial problems*, 23(1):85–103, 1972.
- 15 Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Fixed-parameter tractable algorithm and polynomial kernel for max-cut above spanning tree. *Theory of Computing Systems*, 64(1):62–100, 2020.
- 16 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.
- 17 Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.
- 18 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

# Cryptanalysis of a Generalized Subset-Sum Pseudorandom Generator

Charles Bouillaguet ✉

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Florette Martinez ✉

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Damien Vergnaud ✉

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

---

## Abstract

We present attacks on a generalized subset-sum pseudorandom generator, which was proposed by von zur Gathen and Shparlinski in 2004. Our attacks rely on a sub-quadratic algorithm for solving a vectorial variant of the 3SUM problem, which is of independent interest. The attacks presented have complexities well below the brute-force attack, making the generators vulnerable. We provide a thorough analysis of the attacks and their complexities and demonstrate their practicality through implementations and experiments.

**2012 ACM Subject Classification** Security and privacy → Cryptography

**Keywords and phrases** Cryptography, pseudo-random generator, subset-sum problem, 3SUM problem, cryptanalysis

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.23

**Supplementary Material** *Software*: <https://github.com/floretteM/Knapsack>  
archived at `swh:1:dir:dd38a4a2c04ee30599a006d1882aa1975d4fc778`

**Funding** This work was supported in part by the French Agence Nationale de la Recherche under project “GORILLA” (ANR-20-CE39-0002). Preliminary computational experiments were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

**Acknowledgements** The authors are grateful to the anonymous reviewers for their insightful comments and valuable suggestions.

## 1 Introduction

A pseudo-random number generator is a deterministic algorithm that runs in polynomial time using a short random seed as its input, and produces a long sequence that is indistinguishable from a truly random sequence in polynomial time. The versatile applications of pseudo-random numbers have been extensively explored in the literature, particularly in cryptography where they are employed for tasks such as key generation, encryption, and digital signatures.

In 1985, Rueppel and Massey introduced the *knapsack generator* (or *subset sum generator*) [16] whose security ultimately relies on the NP-hard *modular subset sum problem*: given integers  $\omega_0, \dots, \omega_{n-1}$ ,  $t$  and  $q$ , find a subset of the  $\omega_i$ ’s that sum to  $t$  modulo  $q$ , i.e. to find bits  $x_0, \dots, x_{n-1} \in \{0, 1\}$  such that

$$\sum_{i=0}^{n-1} x_i \omega_i = t \pmod{q}.$$



© Charles Bouillaguet, Florette Martinez, and Damien Vergnaud;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 23; pp. 23:1–23:15

Leibniz International Proceedings in Informatics



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

In the *knapsack generator*, the modulus  $q$  is usually taken as a power of 2,  $q = 2^n$ , the *weights*  $\omega_0, \dots, \omega_{n-1}$  are kept secret and given  $n$  secret *control bits*  $u_0, \dots, u_{n-1}$ , one extends them using a linear feedback generator (a fast but non-cryptographically secure pseudo-random number generator) to obtain a flow of pseudo-random bits  $(u_i)_{0 \leq i \leq N+n-2}$ . For  $i \in \{0, \dots, N-1\}$ , one then computes

$$v_i = \sum_{j=0}^{n-1} u_{i+j} \omega_j \bmod 2^n$$

and outputs  $y_i$  which are the  $\rho = n - \ell$  leading bits of  $v_i$  where  $\ell$  is a given parameter.

In 2011, Knellwolf and Meier [12] presented a cryptanalysis of this generator. They used a *guess-and-determine* strategy coupled with lattice-based techniques to recover most of the key in relevant instances of the generator. In order to run said attack, they needed to guess all the  $n$  initial control bits. Hence their attack had a time complexity  $\Omega(2^n)$ . In 2009, Von zur Gathen and Shparlinski [20] presented the *fast knapsack generator* that had a far smaller key and was sensibly faster but had not undergone a serious cryptanalysis until recently [14]. We consider another variant of the subset sum pseudorandom generator, suggested by von zur Gathen and Shparlinski in 2004 [19].

The family of generators proposed by von zur Gathen and Shparlinski can be described in an abstract way using two integer parameters  $\lambda$  and  $n$  and three independent components:

- a control-sequence generator  $\text{CSG} : \{0, 1\}^\lambda \times \mathbb{N} \rightarrow \{0, 1\}^n$ ;
- an Abelian cyclic group  $(\mathbb{G}, +)$  of order  $q$  where the group law is denoted additively;
- a deterministic and public conversion function  $\Psi : \mathbb{G} \rightarrow \{0, 1\}^\rho$  where  $\rho$  denotes the output length of the pseudo-random generator.

The seed of this generalized subset-sum generator consists in a bit-string  $\text{seed}_0 \in \{0, 1\}^\lambda$  and  $n$  group elements  $P_1, \dots, P_n \in \mathbb{G}$ . The bit size of the seed is thus equal to  $\lambda + n \cdot \lceil \log_2(q) \rceil$ . At each iteration  $i \in \mathbb{N}$ , the control-sequence generator generates an  $n$ -bit string  $\mathbf{v}_i = (v_i^1, \dots, v_i^n) = \text{CSG}(\text{seed}_0, i)$ , computes the group element  $Q_i$  defined by

$$Q_i = [v_i^1]P_1 + \dots + [v_i^n]P_n \in \mathbb{G}$$

and outputs  $s_i = \Psi(Q_i) \in \{0, 1\}^\rho$ .

In the Rueppel-Massey classical subset sum generator, the group  $\mathbb{G}$  is thus the group of modular residue  $\mathbb{G} = \mathbb{Z}_q$ , the control-sequence generator is defined by a linear feedback shift register and the conversion function is a truncation. In [19], von zur Gathen and Shparlinski proposed to use for  $\mathbb{G}$  the group of rational points of an elliptic curve defined over a (prime) finite field, a linear feedback shift register as the control-sequence generator and again a truncation for the conversion function (more precisely, truncation of the  $x$ -coordinate of the elliptic curve point  $Q_i$ ). They proposed to use  $\lambda = n$  and an elliptic curve defined over a finite field  $\mathbb{Z}_p$  where  $p$  is a  $n$ -bit prime number. By the Hasse-Weil theorem [9], the number of group elements  $q$  is around  $2^n$  and the total seed size is  $\simeq n + n \cdot n = n \cdot (n + 1)$ . They suggested that  $\Psi$  should discard  $\ell = \log_2(n)$  low-order bits of the  $x$ -coordinate of the point before using it as pseudo-random output.

Von zur Gathen and Shparlinski claimed that: “the only available attack on this generator is the brute force search over all parameters defining this generator” and thus using  $n$  as small as 12 should provide a 128-bit security level. The statistical properties of the sequences generated by this pseudo-random generator were analyzed in [4, 1, 8] but its cryptographic security has not been studied up to the present article. We present a simple attack against this generator and a lattice-based attack on another variant derived from this abstraction. In the instantiation suggested by von zur Gathen and Shparlinski, our attack

has complexity  $\mathcal{O}(2^{1.78n})$  well below the  $\mathcal{O}(2^{n(n+1)})$  brute-force attack. Our attacks rely on a sub-quadratic algorithm for solving a vectorial variant of the 3SUM problem, which is of independent interest. We provide a thorough analysis of the attacks and their complexities, and demonstrate their practicality through implementations and experiments.

## 2 High-level description of the attack

We consider the case where the control sequence generated by the CSG is known by the adversary. If this is not the case, they can simply try all possible values for  $\text{seed}_0 \in \{0, 1\}^\lambda$  which increases the complexity of the attack by a multiplicative factor  $2^\lambda$ .

We assume that the control sequence generator outputs uniform and independent  $n$ -bit strings  $\mathbf{v}_i = \text{CSG}(\text{seed}_0, i)$  for each  $i \in \mathbb{N}$ . Note that in the concrete schemes that we attack, this is obviously false; we nevertheless carry our analysis under this assumption and our experimental results will show that it actually holds in practice.

Let us suppose that an adversary finds three indices  $i, j, k$  such that  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$  as vectors of integers, i.e. where the addition is performed coordinate-wise over  $\mathbb{Z}$ . In this case, the adversary knows that the relation  $Q_i + Q_j = Q_k$  holds in the group  $\mathbb{G}$ . They are not given the actual values of the points  $Q_i, Q_j$  and  $Q_k$  but only the values  $\Psi(Q_i), \Psi(Q_j)$  and  $\Psi(Q_k)$ . Assume that the number of preimages through  $\Psi$  is limited and that the adversary can efficiently compute them; they can simply check if  $\Psi(X + Y) = \Psi(Q_k)$  for all  $(X, Y)$  such that  $\Psi(X) = \Psi(Q_i)$  and  $\Psi(Y) = \Psi(Q_j)$ . If there exists only one such pair  $(X, Y)$  then the adversary can safely assume that  $Q_i = X, Q_j = Y$  (and  $Q_k = X + Y$ ).

The number of pairs  $(X, Y)$ 's such that

$$\Psi(X + Y) = \Psi(Q_k) \tag{1}$$

is difficult to estimate and depends heavily on the group  $\mathbb{G}$  and the conversion function  $\Psi$ . In [17], Shoup studied the computational complexity of the discrete logarithm in Abelian groups in the context of algorithms which do not exploit any special properties of the encodings of group elements. Shoup introduced the *generic group model* where each group element is encoded as a unique and arbitrary binary string (picked uniformly at random and independent of the actual group structure). As a consequence, it is not possible for an algorithm in this model to exploit any special properties of the encodings and group elements can only be operated on using an oracle that provides access to the group operations. If we make a similar assumption on the group  $\mathbb{G}$  and the conversion function  $\Psi$  is a truncation of  $\ell$  bits out of the  $(\log_2 q)$ -bit encodings of group elements, then we can expect that the number of preimages is close to  $2^\ell$  and the number of pairs  $(X, Y)$  different from  $(Q_i, Q_j)$  for which (1) holds is expected to be

$$2^\ell \cdot 2^\ell / 2^{\log_2(q) - \ell} \simeq 2^{3\ell} / q. \tag{2}$$

In particular if  $\rho > 2 \cdot \log_2(q) / 3$ , one expects the number of candidates for  $(Q_i, Q_j, Q_k)$  to be constant in a “generic” group. It is worth mentioning that this assumption does not hold in the classical knapsack generator that uses the group  $\mathbb{G} = \mathbb{Z}_m$  since in this case, the number of candidates for a single equation will be about  $2^{2\ell}$ .

Each relation  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$  gives two relations in the group  $\mathbb{G}$ :

$$Q_i = v_i^1 P_1 + \dots + v_i^n P_n \quad \text{and} \quad Q_j = v_j^1 P_1 + \dots + v_j^n P_n$$

If the adversary collect sufficiently many linearly independent such relations, they would be able to retrieve all the weights used in the generalized knapsack generator.

■ **Table 1** Tabulating all solutions of  $x + y = z$  for  $x, y, z \in \{0, 1\}$ .

$x$	0	0	0	0	1	1	1	1
$y$	0	0	1	1	0	0	1	1
$z$	0	1	0	1	0	1	0	1
$x + y$	0	0	1	1	1	1	2	2

In the following, we describe and analyse an algorithm to find “good triplets” of indices  $(i, j, k)$  such that  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$  and show how to use it to attack the elliptic knapsack generator when  $\rho = n - \log_2(n)$  (as suggested by von zur Gathen and Shparlinski).

### 3 Finding “Good Triplets”

Assume that three lists  $A, B$ , and  $C$ , each of size  $N$ , are made of uniformly random  $n$ -bit strings. Let  $Y$  be the random variable that counts the number of triplets  $(x, y, z) \in A \times B \times C$  such that  $x + y = z$  when  $x, y$  and  $z$  are seen over  $\mathbb{Z}^n$  and not modulo 2. When this relation holds, we call  $(x, y, z)$  a “good triplet”. Our goals in this section are twofold: 1) lower-bound the probability that  $A, B$  and  $C$  contain a good triplet and 2) design an algorithm to find good triplets efficiently.

As a warm-up, examining the simplest case ( $n = 1$ ) is interesting (cf. Table 1). Looking at this table, we see that  $\Pr(x + y = z) = 3/8$ . We next prove the following

► **Theorem 1.**  $E(Y) = N^3 \left(\frac{3}{8}\right)^n$ , and  $\Pr(Y = 0) \leq \frac{1}{N^3} \left(\frac{8}{3}\right)^n + \frac{3}{N} \left(\frac{10}{9}\right)^n + \frac{3}{N^2} \left(\frac{4}{3}\right)^n$ .

The proof is given in Appendix A. It boils down to estimating the variance of  $Y$  and using the second-moment inequality. With  $N = \alpha(8/3)^{n/3}$ , Theorem 1 yields:

$$\Pr(Y = 0) \leq \frac{1}{\alpha^3} + \frac{3}{\alpha}(0.801\dots)^n + \frac{3}{\alpha^2}(0.69\dots)^n.$$

Therefore, setting  $\alpha = 10$  is sufficient to ensure that a good triplet exists with probability 99.9%. In addition, it follows from the theorem that

$$\Pr(Y = 0) \leq \frac{1}{E(Y)} + \frac{3}{(E(Y))^{2/3}} + \frac{3}{(E(Y))^{1/3}}. \quad (3)$$

#### 3.1 A Simple Sub-Quadratic Algorithm to Find Good Triplets

Finding a “good triplet” (such that  $x + y = z$ ) can be done using a naive quadratic algorithm: for all pairs  $(x, y)$  in  $A \times B$ , check if  $x + y \in C$ ; if so, return  $(x, y, x + y)$ ; after this loop, return  $\perp$  (to handle the case where the algorithm fails). This could potentially be sped up a little by exploiting the fact that  $x$  and  $y$  are necessarily disjoint.

In this section, we present a simple algorithm to find a good triplet more efficiently. We work under the assumption that the input lists have size  $N := \alpha(8/3)^{n/3}$  for some constant  $\alpha \geq 4$ . Under this condition, (3) ensures that there is a good triplet with probability at least  $\frac{3}{64}$ . This assumption will be relaxed in the next section.

Looking again at Table 1, we see that  $\Pr(x = 1 \mid x + y = z) = 1/3$  while  $\Pr(z = 1 \mid x + y = z) = 2/3$ . In other terms, even though  $x, y, z$  are sampled uniformly at random, if we restrict our attention to good triplets, then  $x$  and  $y$  are biased towards zero (sparse) while  $z$  is biased towards 1 (dense).

This observation suggests an algorithm to find good triplets efficiently: remove from  $A, B$  (resp.  $C$ ) input vectors of Hamming weight different from  $n/3$  (resp.  $2n/3$ ), then run the naive quadratic algorithm on what remains.

► **Theorem 2.** *This algorithm terminates in  $\mathcal{O}(N^e)$  with  $e = 2\ln(9/4)/\ln(8/3) \approx 1.654$  and succeeds with probability  $\Omega(\frac{1}{n})$ .*

**Proof.** Let  $H$  denote the binary entropy function, meaning that  $H(x) = -x \log_2(x) - (1-x) \log_2(1-x)$ , for all  $0 < x < 1$ . The following standard bounds for the binomial coefficient can be derived from Stirling's formula:

$$\frac{2^{nH(x)}}{\sqrt{8nx(1-x)}} \leq \binom{n}{xn} \leq \frac{2^{nH(x)}}{\sqrt{2\pi nx(1-x)}}, \quad (0 < x < 1/2) \quad (4)$$

It follows from the discussion just before the statement of the theorem that there are  $3^n$  good triplets on  $n$  bits (out of  $8^n$  triplets in total). The number of good triplets that satisfy the weight condition imposed by the algorithm is

$$N = \binom{n}{n/3, n/3, n/3} = \binom{n}{2n/3} \binom{2n/3}{n/3} \geq \frac{2^{nH(2/3)}}{\sqrt{n}4/3} \frac{2^{2n/3}}{2\sqrt{n/3}} = \frac{3\sqrt{3}}{8n} 3^n.$$

If the input list contain a good triplet, then the algorithm described above returns it with probability greater than  $0.65/n$ . The claimed time complexity is in fact a consequence of the *next* theorem (Theorem 5), and we will therefore not prove it here. ◀

### 3.2 Sub-Quadratic Algorithm with Overwhelming Success Probability

We generalize the algorithm of the previous section by relaxing the weight condition. This yields Algorithm 1. It takes an additional argument  $w$  controlling the maximum allowed weight. In the sequel, all the stated complexities must be understood “up to a constant factor”. Let  $\epsilon$  denote a constant in the open interval  $(0; \frac{1}{6})$ . Let  $X \sim \mathcal{B}(n, p)$  be a binomial random variable. We will use the classical inequality (5) given below, a proof of which can be found in [2] amongst others. Here,  $D(a, p)$  is the Kullback-Leibler divergence between an  $a$ -coin and a  $p$ -coin:

$$\begin{aligned} \Pr(X \leq an) &\leq \exp(-nD(a, p)) && \text{if } a < p. \\ \Pr(X \geq an) &\leq \exp(-nD(a, p)) && \text{if } a > p, \\ D(a, p) &= a \ln \frac{a}{p} + (1-a) \ln \frac{1-a}{1-p}. \end{aligned} \quad (5)$$

We denote by  $\text{wt}(x)$  the Hamming weight of a bit string  $x$ .

■ **Algorithm 1** Find good triplets.

---

```

1: function FINDTRIPLET( $A, B, C, w$ )
2:    $A' \leftarrow \{x \in A \mid \text{wt}(x) \leq w\}$ 
3:    $B' \leftarrow \{y \in B \mid \text{wt}(y) \leq w\}$ 
4:   for all  $x, y \in A' \times B'$  do
5:     if  $x + y \in C$  then
6:       return  $(x, y, z)$ 
7:   return  $\perp$ 

```

---

► **Lemma 3.** *With  $w = n(\frac{1}{3} + \epsilon)$ , if the input contains a good triplet, then Algorithm 1 returns  $\perp$  with probability less than  $2 \exp(-2n\epsilon^2)$ .*

**Proof.** Assume that the input lists contain a good triplet  $(x^*, y^*, z^*)$ . It will be discarded if and only if the weight of either  $x^*, y^*$  is greater than  $w$ . We know that the weight of  $x^*$  and  $y^*$  follows a binomial distribution of parameters  $(n, 1/3)$ , therefore (5) shows that either has weight greater than  $n(1/3 + \epsilon)$  with probability less than  $\exp(-nD(1/3 + \epsilon, 1/3))$ .

The (“well-known”) fact that  $D(p + \epsilon, p) \geq 2\epsilon^2$  combined with union bound (for  $x^*$  and  $y^*$ ) then yields the announced result. ◀

► **Lemma 4.** *Let  $T$  denote the running time of Algorithm 1 with  $w = n(\frac{1}{3} + \epsilon)$ . Then  $E[T] \leq N + N^2 \exp[-2nD(\frac{1}{3} + \epsilon, \frac{1}{2})]$ .*

**Proof.** Filtering the input lists and keeping only low-weight vectors can be done in linear time. Given the complexity of the naive quadratic algorithm, the total time complexity is simply  $T = N + |A'| \cdot |B'|$ .

Let  $X \sim \mathcal{B}(n, 1/2)$  be a binomial random variable modeling the weight of a random  $n$ -bit vector. Such a vector belongs to  $A'$  or  $B'$  if its weight is less than or equal to  $w$ , and this happens with probability  $s := \Pr(X \leq w)$ . The binomial tail bound (5) yields  $s \leq \exp[-nD(\frac{1}{3} + \epsilon, \frac{1}{2})]$ .

The sizes of  $A'$  and  $B'$  are stochastically independent random variables following a binomial distribution of parameters  $(N, s)$  with expectation  $Ns$ . The expected running time of the quadratic algorithm on  $A'$  and  $B'$  is therefore  $E[|A'| \times |B'|] = E|A'| \times E|B'| = N^2 s^2$ . Combining this with the upper bound on  $s$  gives the announced result. ◀

► **Theorem 5.** *Write  $e = 2 \cdot \frac{\ln(9/4)}{\ln(8/3)} \approx 1.654$ . For all  $d > e$  there is an algorithm that runs in time  $\mathcal{O}(N^d)$ , where  $N$  denotes the size of the input list and fails to reveal a good triplet present in the input with negligible probability (in  $n$ ).*

**Proof.** Let  $e < d < 2$  be a complexity exponent greater than the bound  $e$  given in the statement of the theorem. There always exist  $\epsilon > 0$  such that

$$d = 2 - 6 \frac{D(\frac{1}{3} + \epsilon, \frac{1}{2})}{\frac{1}{3} \ln \frac{8}{3} + \epsilon \ln 2}.$$

Indeed, setting  $\epsilon = 0$  in this expression yields the lower-bound exponent  $e$  of the theorem, and the expression of  $d$  is increasing as a function of  $\epsilon$ ; it reaches  $d = 2$  for  $\epsilon = 1/6$ .

Let  $N_0 := (8/3)^{n/3}$ , so that input lists of size  $N_0$  contain a single good triplet in average. We distinguish two cases depending of the size of the input lists.

Suppose that  $N \leq 2^{\epsilon n} N_0$ , where  $N$  denotes the size of the input lists. In this case run Algorithm 1 with  $w = n(\frac{1}{3} + \epsilon)$ . Lemma 3 guarantees the exponentially small failure probability while lemma 4 tells us that the expected running time  $T$  is less than  $N + N^2 \exp[-2nD(\frac{1}{3} + \epsilon, \frac{1}{2})]$ .

A quick calculation shows that the algorithm then runs in time  $\mathcal{O}(N^d)$  – the value of  $d$  has been chosen for this purpose. The theorem is proved in this case.

If  $N > 2^{n\epsilon} N_0$ , then slice the input lists in chunks of size  $4N_0$  and run Algorithm 1 with  $w = n/3$  on each successive chunk until a solution is found. Each chunk contains a good triplet with probability at least  $\frac{3}{64}$  thanks to (3). The algorithm reveals this triplet, if it exists, with probability  $\Omega(\frac{1}{n})$ , because it always works if the algorithm of the previous section works.

There are  $2^{en}/4$  chunks (i.e., exponentially many). Because the chunks are disjoint parts of the input lists, success in a chunk is independent from the others. Therefore the probability that this process fails to reveal a good triplet is negligible. The running time of this procedure is  $\mathcal{O}(NN_0^{e-1})$ . Because  $N_0 \leq N$ , this is less than  $\mathcal{O}(N^e)$ . ◀

► **Remark 6.**  $A_b, B_b$  and  $C_b$  the subsets of strings of  $A, B$  and  $C$  whose first bit is equal to  $b$  (for  $b \in \{0, 1\}$ ), then a good triplet necessarily belongs to one of the three sets  $A_0 \times B_0 \times C_0$ ,  $A_0 \times B_1 \times C_1$  or  $A_1 \times B_0 \times C_1$  (and the search for a good triplet in  $A \times B \times C$  thus reduces to the search in those three sets). The expected cardinality of  $A_0, A_1, B_0, B_1, C_0$  and  $C_1$  is  $N/2$  and applying this idea recursively, one obtains (assuming the division is always done in a “balanced” manner) a time complexity  $T(N)$  which heuristically satisfies the recursion  $T(N) = 3T(N/2) + \mathcal{O}(N)$  (and thus  $T(N) = \mathcal{O}(N^{\log_2(3)}) = \mathcal{O}(N^{1.59})$ ). This improves a bit the complexity of our algorithm, but this “divide-and-conquer” approach is probably more complex to implement. It would be interesting to see if one can combine this approach with our filtering technique.

#### 4 Practical Key-recovery Attack on the von zur Gathen-Shparlinski Elliptic Subset Sum Generator

In this section, we consider the instantiation of the knapsack generator suggested by von zur Gathen and Shparlinski in [19]. In particular, the group  $\mathbb{G}$  is an elliptic curve  $E$  defined over a (prime) finite field  $\mathbb{F}_p$  (where  $p \geq 5$  is an  $n$ -bit prime number). It is a rational curve given by the following Weierstrass equation

$$E : y^2 = x^3 + ax + b$$

for some  $a, b \in \mathbb{F}_p$  with  $4a^3 + 27b^2 \neq 0$ . It is well known that the set  $E(\mathbb{F}_p)$  of  $\mathbb{F}_p$ -rational points (including the special point  $O$  at infinity) forms an Abelian group with an appropriate composition rule (denoted additively) where  $O$  is the neutral element – for more details on elliptic curves, we refer to [5, 21]. Von zur Gathen and Shparlinski suggested to use a conversion function  $\Psi : E \rightarrow \{0, 1\}^\rho$  that simply truncates  $\ell = \log_2(n)$  least significant bits of the x-coordinate of a point (with  $\rho = n - \ell$ ). An  $n$ -bit linear feedback shift register is used as the control-sequence generator (as in the Rueppel-Massy classical knapsack generator) and the overall seed length is thus  $n(n + 1)$  bits.

##### 4.1 Attack on the Elliptic Subset Sum Generator

The adversary first “guesses”  $\text{seed}_0$ . In other terms, all subsequent steps have to be repeated  $2^n$  times, one for each possible value of  $\text{seed}_0$ .

Following the analysis from Section 3, one needs to construct three sets  $A, B, C$  of independent  $n$ -bit strings size  $N = \alpha(8/3)^{n/3} \leq \alpha 2^{0.472n}$  in order to find a good triplet  $(i, j, k)$  such that  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$  time  $\mathcal{O}(N^{1.654\dots}) = \mathcal{O}(2^{0.78n})$  with probability at least  $1 - 7/\alpha$ . We need to have  $n/2$  such good triplets in order to find the  $n$  points  $P_1, \dots, P_n$  used as weights in this elliptic knapsack generator, and we can hope to obtain them with constant positive probability from an output sequence made of  $\Omega(n^2(8/3)^{n/3})$  output values in  $\{0, 1\}^\rho$ . In our implementation, we do not distinguish the sets  $A, B$ , and  $C$  and simply run the algorithm from the previous section with  $A = B = C$  the sets of all vectors  $\mathbf{v}_i$  corresponding to all known outputs  $s_i \in \{0, 1\}^\rho$ .

Note that, as in the classical knapsack generator, the control sequence is not made of independent  $n$ -bit strings since if one denotes  $(u_n)_{n \geq 0}$  the sequence output by the linear feedback shift register, we have

$$\mathbf{v}_i = (v_i^1, \dots, v_i^n) = (u_i, u_{i+1}, \dots, u_{i+n-1}) \in \{0, 1\}^n$$

for  $i \in \mathbb{N}$ . The analysis given in Section 3 does not apply to such sequences but we make the heuristic assumption that these  $n$ -bit tuples are “sufficiently” random and that our algorithm will succeed with a similar probability (this heuristic is validated by our experiments).

We then follow the general idea given above but for each good triplet  $(i, j, k)$  such that  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$ , if the adversary finds two points  $X$  and  $Y$  on the elliptic curve such that  $\Psi(X) = s_i$ ,  $\Psi(Y) = s_j$  and  $\Psi(X + Y) = s_k$ , then this gives rise to two possible relations:

1.  $X = Q_i$ ,  $Y = Q_j$  (and  $X + Y = Q_i + Q_j = Q_k$ ), but also
2.  $X = -Q_i$ ,  $Y = -Q_j$  (and  $X + Y = -(Q_i + Q_j) = -Q_k$ ).

This is due to the fact that on an elliptic curve, a point and its negative have representations with much in common since they share the same the x-coordinate (and the y-coordinates are opposites). This “non-genericness” of elliptic curves is well-known and has important consequences in cryptography<sup>1</sup>. However, with a truncation of  $\log_2(n)$  bits of the x-coordinate of the points, we expect the number of points triple compatible with  $(s_i, s_j, s_k)$  to be equal to only 2.

Note that for the first good triplet computed in the attack, the sign is not a problem since the generator parametrized with the  $n$  points  $P_1, \dots, P_n$  outputs the same sequence as the one parametrized with the  $n$  points  $-P_1, \dots, -P_n$ . The adversary can then pick up arbitrarily  $(Q_i, Q_j) = (X, Y)$  or  $(Q_i, Q_j) = (-X, -Y)$ . However, for the subsequent relations obtained from other good triplets, the sign choice may be incompatible with the first one and this will result in a system with no solutions. In order to be able to solve the system, we need to have  $n$  linear relations among the points  $P_1, \dots, P_n$  and each good triplet gives us two such relations (the third one is by construction is a linear combination of the two others and is useless in solving the linear system). Assuming that  $n$  is even, one needs to make  $n/2 - 1$  choices for the sign of each relation (after the first one), and the adversary can simply “guess” all such signs. This multiplies the running time of the algorithm by a factor  $2^{n/2-1}$ .

Once the  $n/2$  good triplets have been found, the algorithm can inverse the system and obtain the  $n$  points  $P_1, \dots, P_n$ . From these values, the points  $P_1, \dots, P_n$ . The overall complexity of the attack is thus

$$\mathcal{O}\left(2^n(2^{0.78n} + 2^{2\log(n)} + 2^{n/2-1}) + \text{poly}(n)\right) = \mathcal{O}\left(2^{1.78n}\right)$$

binary operations.

## 4.2 Experimental Results

We implement our attack using sagemath v.9.5 on a laptop. Our codes are available at <https://github.com/floretteM/Knapsack>. In our implementation, we did not need the exhaustive search on the signs mentioned above since we instead looked for good triplets that involve a point already found (and this sets up the sign with certainty). This makes the probability of finding such good triplets more complex to analyse but this trick works well in practice.

We first consider the elliptic curve defined by the equation  $y^2 = x^3 + 5x + 5$  over  $\mathbb{F}_p$  where  $p = 2^{16} - 15$ . This curve contains  $q = 65111$  points. We present the attack when the control sequence  $(\mathbf{v}_i)$  is known and we consider  $n = 16$  as suggested by von zur Gathen and Shparlinski. The key size in this setting is equal to 256 bits. We present in Table 2 the number  $m$  of outputs needed and the time necessary to recover the secret weights with probability at least 50% when  $\ell$  bits are missing.

<sup>1</sup> For instance, the ECDSA signature scheme is malleable in the sense that if the pair of integers  $(r, s)$  is

■ **Table 2** Key-recovery with exhaustive search and  $q$  a 16-bit integer.

$\ell$	1	2	3	4	5	6
$m$	1000	1000	1000	1000	1000	1885
time	6.9s	5.3s	5.6s	5.02s	5.7s	26.7s

When 7 bits are truncated we cannot recover the weights even with 3000 outputs. But we earlier observed that the algorithm would not work well if  $\ell > \log_2(q)/3$ , see (2). With the proposed choice of  $\ell = \log_2 n = 5$ , our results are coherent with the heuristic.

To test the limits of our attack, we also implement it for elliptic curves with larger group orders (*i.e.* for parameters larger than those suggested by von zur Gathen and Shparlinski). This gives an algorithm with overall complexity  $\mathcal{O}(2^n(2^{0.78n} + 2^{n/2}) \cdot \text{poly}(n, \log(p)))$ .

We consider the elliptic curve defined by the equation  $y^2 = x^3 + x + 14$  over  $\mathbb{F}_p$  where  $p = 2^{40} + 15$  but still  $n = 16$ . With this choice we can focus on recovering the points of the elliptic curves from the outputs without being bothered by finding the good triplets. This curve contains  $q = 1099510687747$  points. We present in Table 3 the number  $m$  of outputs needed and the time necessary to recover the secret weights with probability at least 50% when  $\ell$  bits are missing.

■ **Table 3** Key-recovery with exhaustive search and  $q$  a 40-bit integer.

$\ell$	1	2	3	4	5	6	7	8	9
$m$	1885	1885	1885	1885	1885	1885	1885	1885	1750
time	2.1s	2.1s	2.08s	2.5s	2.6s	2.1s	3.5s	8.3s	26.7s

## 5 Practical Key-recovery Attack on the Subset Product Generator

Following the generalization of the knapsack generator to elliptic curves proposed by von zur Gathen and Shparlinski, it is natural to consider other variants using Abelian groups of interest in cryptography. The most natural choice is to use (a subgroup of) the multiplicative group of a finite field  $\mathbb{Z}_p$  for some prime number  $p$ . This group is certainly not generic since there exist sub-exponential time discrete logarithm algorithms in these groups, but it seems that representation of group elements by the unique member of its class in  $\{0, \dots, p-1\}$  is sufficiently “generic” that using truncation of their bit-representation as a conversion function would permit an adversary to mount a lattice-based attack on this generator even if a quarter of the bits of each group elements is discarded when computing the output of the generator.

More precisely, in this section, we consider a multiplicative variant of the subset sum generator where:

- the control-sequence generator is a linear feedback shift register with a  $\lambda$ -bit seed;
- the Abelian cyclic group  $(\mathbb{G}, \cdot)$  is the multiplicative group of a (prime) finite field  $\mathbb{Z}_p$  (note that it is denoted multiplicatively);
- the public conversion function  $\Psi : \mathbb{G} \rightarrow \{0, 1\}^\rho$  where  $\rho = \lfloor \alpha \cdot \log_2(p) \rfloor$  is simply the truncation of  $\lceil (1 - \alpha) \log_2(p) \rceil$  bits of the unique member of its group element class in  $\{0, \dots, p-1\}$ . The notation  $\div$  denotes the euclidean division.

---

a valid signature of a given message then so is  $(r, -s)$  [18].

We call this generator the *subset product generator*.

### 5.1 Description of the Attack

In this setting, the seed consists in a bit-string  $\text{seed}_0 \in \{0, 1\}^\lambda$  and  $n$  group elements  $g_1, \dots, g_n \in \mathbb{Z}_p^*$ . The bit size of the seed is thus equal to  $\lambda + n \cdot \lceil \log_2(p) \rceil$ . At each iteration  $i \in \mathbb{N}$ , the control-sequence generator generates an  $n$ -bit string  $\mathbf{v}_i = (v_i^1, \dots, v_i^n) = \text{CSG}(\text{seed}_0, i)$ , computes the group element  $h_i$  defined by

$$h_i = g_1^{v_i^1} \cdots g_n^{v_i^n} \in \mathbb{Z}_p^*$$

and outputs  $s_i = \Psi(h_i) = h_i \text{div } 2^\ell \in \{0, 1\}^k$  where  $p$  is a  $(k + \ell)$ -bit long prime number (with  $k = \lfloor \alpha \cdot \log_2(p) \rfloor$ ).

A straightforward adaptation of the attack of the Section 4 gives an attack with complexity  $O(2^\lambda \cdot (2^{0.78n} + p^{2(1-\alpha)}))$  for  $\alpha \geq 2/3$ . Note that the complexity does not involve the  $O(2^{n/2})$  term that came from the indecision on the signs in the elliptic curve variant of the knapsack generator. We remark that one can improve the complexity of the attack by replacing the brute-force search on the missing bits with the use of Coppersmith technique to retrieve them.

**Coppersmith's method.** Coppersmith's method [7, 6] is a technique to find small integer zeroes of univariate or bivariate polynomials modulo a given integer. It has been generalized for finding small roots of (modular) multivariate polynomial equations with integer coefficients by several authors and notably used to attack algebraic pseudo-random generators (see [11, 10, 3, 14] and references therein). These techniques work by constructing a Euclidean lattice associated with the system of equations, and then finding short vectors in this lattice using lattice reduction algorithms. In its most basic variant, given a polynomial  $f(X_1, \dots, X_k)$  defined modulo an integer  $p$ , one can find a "small" root  $(x_1, \dots, x_k) \in \mathbb{Z}_p^k$  under the condition that  $|x_i| \leq B_i$  for some bounds  $(B_1, \dots, B_k)$ . The method succeeds (heuristically) in polynomial time when (up to small constant factors),

$$\prod_{i \in M} B_1^{i_1} \cdots B_k^{i_k} \leq p$$

when  $f$  can be written as a sum of monomials of the form

$$f(X_1, \dots, X_k) = \sum_{i \in M} a_i X_1^{i_1} \cdots X_k^{i_k}$$

for some  $a_i \in \mathbb{Z}_p^*$ . For this simple variant, the lattice is constructed using only the polynomial  $f$  but there exist variants – with better upper-bounds on the root  $(x_1, \dots, x_k)$  – using lattices of higher dimensions with shifts or powers of the polynomial  $f$  (see [11] for details).

**Description of the attack.** For a vector  $\mathbf{v}_i$  output by the control sequence generator, we have

$$h_i = g_1^{v_i^1} \cdots g_n^{v_i^n} \in \mathbb{Z}_p^*$$

with  $h_i = (2^\ell s_i + x_i)$  where  $x_i \in \{0, \dots, 2^\ell - 1\}$  is some value unknown to the adversary. Given a good triplet  $(i, j, k)$  with  $\mathbf{v}_i + \mathbf{v}_j = \mathbf{v}_k$ , we have  $h_i \cdot h_j = h_k \pmod p$  and thus:

$$(2^\ell s_i + x_i) \cdot (2^\ell s_j + x_j) = (2^\ell s_k + x_k) \pmod p.$$

The unknowns  $(x_i, x_j, x_k)$  are thus “small” roots of an equation of the form

$$Ax_i + Bx_j + x_ix_j - x_k + C = 0 \pmod{p}$$

where  $A = 2^\ell s_i$ ,  $B = 2^\ell s_j$  and  $C = (2^\ell s_i \cdot 2^\ell s_j - 2^\ell s_k) \pmod{p}$  are values known by the adversary. One can thus apply Coppersmith’s technique to this polynomial and the basic technique (without using shifts or powers of the polynomial) will succeed if  $|x_i|, |x_j|, |x_k| \leq p^{1/5}$ . A simple trick allows us to improve readily this bound by setting  $y = x_ix_j - x_k$  such that  $|y| \leq 2^{2\ell}$  and solving the equation

$$g(x_i, x_j, y) = Ax_i + Bx_j + y + C = 0 \pmod{p}$$

in  $(x_i, x_j, y)$  is sufficient to recover  $(x_i, x_j, x_k)$ . Using the basic Coppersmith’s technique (again without using shifts or powers of this polynomial), this attack will succeed (heuristically) in polynomial-time if  $|x_i|, |x_j|, |x_k| \leq p^{1/4}$ . For  $\alpha \geq 3/4$ , we thus obtain an attack with the overall complexity

$$\mathcal{O}(2^\lambda \cdot 2^{0.78n} + n \cdot \text{poly}(\log_2(p))) = \mathcal{O}(2^\lambda \cdot 2^{0.78n}).$$

► **Remark 7.** Note that we can improve the bound on the size of the “small” root by using shifts and powers of the polynomial  $g(x_i, x_j, y)$ . For instance, if one considers the family of four polynomials

$$\{g, x_i \cdot g, x_j \cdot g, g^2\}$$

that vanishes in  $(x_i, x_j, y)$  modulo  $p$  with total multiplicity  $(1 + 1 + 1 + 2) = 5$  and involve the following set of monomials:

$$\{x_i, x_j, y, x_i^2, x_ix_j, x_iy, x_j^2, x_jy, y^2\}$$

with a sum of degrees equal to  $(1 + 1 + 2 + 2 + 2 + 3 + 2 + 3 + 4) = 20$ , we obtain that the Coppersmith’s method succeeds (heuristically) if  $|x_i|, |x_j|, |x_k| \leq p^{5/20} = p^{1/4}$  (see [11]). This gives the same bound as above. However, if we reintroduce the variable  $x_k$  and replace the monomial  $x_ix_j$  by  $y + x_k$ , the total degree of the set of monomials decreases to 19 and this decreases the bound to  $p^{5/19}$ . It is possible to decrease a bit further the exponent of  $p$  in this bound, at the cost of using a lattice of higher dimension in Coppersmith’s technique using the technique of unravelled linearization from [10] (see also [3]).

## 5.2 Experimental Results

**Exhaustive search on the truncated bits.** We consider first the finite field  $\mathbb{F}_p$  with  $p = 2q + 1$  and  $q = 99839$ . We choose weights in the cyclic multiplicative group  $\mathbb{G}$  of order  $q$  made by the non-quadratic residues of  $K$  minus zero. We present the attack when the control sequence  $(\mathbf{v}_i)$  is known and we consider  $n = 16$  for which the key size is equal to 256 bits. We present in Table 4 the number  $m$  of outputs needed and the time necessary to recover the secret weights with probability at least 50% when  $\ell$  bits are missing.

When 7 bits are truncated we cannot recover the weights even with 1885 outputs.

Now we consider the finite field  $\mathbb{F}_p$  with  $p = 2q + 1$  and

$$q = 72536599031050480402372360602698911648481683373808860129469667649180998227293$$

a 256-bit number, but still  $n = 16$ . With this choice we can focus on recovering the points from the outputs without being bothered by finding the good triplets.

## 23:12 Cryptanalysis of a Generalized Subset-Sum Pseudorandom Generator

■ **Table 4** Key-recovery with exhaustive search and  $q$  a 16-bit integer.

$\ell$	1	2	3	4	5	6
$m$	1000	1000	1000	1000	1000	1885
time	0.51s	0.45s	0.44s	0.47s	0.58s	2.1s

■ **Table 5** Key-recovery with exhaustive search and  $q$  a 256-bit integer.

$\ell$	1	2	3	4	5	6	7	8	9
$m$	1000	1000	1000	1000	1000	1000	1000	1000	1000
time	0.46s	0.50s	0.48s	0.43s	0.55s	0.70s	0.87s	1.9s	6.6s

**Coppersmith method.** We consider the attack on the second group with  $p = 2q + 1$  and  $q$  a 256-bit number. First, we implement the attack with the single polynomial  $g = Ax_i + Bx_j + y + C$ . As the Coppersmith method is a bit more unpredictable, we present in Table 6 the number  $m$  of outputs needed and the time necessary to recover the weights with probability at least 50% when  $\ell$  bits are missing.

■ **Table 6** Key-recovery with Coppersmith method and  $q$  a 256-bit integer.

$\ell$	2	4	8	16	32	62	63
$m$	1000	1000	1000	1000	1000	1000	1000
time	0.71s	0.67s	0.68s	0.61s	0.63s	0.51s	0.55s

If we follow the heuristic in Coppersmith's method we should be able to retrieve the weights up to  $\ell = 64$  and  $\ell = 64$  is the first instance where the attack stops working. If we try to consider the family of polynomials  $\{g, x_i g, x_j g, y g, g^2\}$  instead the improvement on the upper-bound from  $p^{1/4}$  to  $p^{5/19}$  would not be significant for 256-bit integers.

---

### References

- 1 Omran Ahmadi and Igor E. Shparlinski. Exponential sums over points of elliptic curves. *J. Number Theory*, 140:299–313, 2014.
- 2 R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51(1):125–131, 1989.
- 3 Aurélie Bauer, Damien Vergnaud, and Jean-Christophe Zapolowicz. Inferring sequences produced by nonlinear pseudorandom number generators using Coppersmith's methods. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 609–626, Darmstadt, Germany, May 21–23 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-30057-8\_36.
- 4 Simon R. Blackburn, Alina Ostafe, and Igor E. Shparlinski. On the distribution of the subset sum pseudorandom number generator on elliptic curves. *Unif. Distrib. Theory*, 6(1):127–142, 2011.
- 5 Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. *Elliptic curves in cryptography*, volume 265 of *Lond. Math. Soc. Lect. Note Ser.* Cambridge: Cambridge University Press, 1999.
- 6 Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189, Saragossa, Spain, May 12–16 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68339-9\_16.

- 7 Don Coppersmith. Finding a small root of a univariate modular equation. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT’96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165, Saragossa, Spain, May 12–16 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68339-9\_14.
- 8 Edwin D. El-Mahassni. On the distribution of the elliptic subset sum generator of pseudorandom numbers. *Integers*, 8(1):article a31, 7, 2008.
- 9 Helmut Hasse. Zur Theorie der abstrakten elliptischen Funktionenkörper. III: Die Struktur des Meromorphismenringes. Die Riemannsche Vermutung. *J. Reine Angew. Math.*, 175:193–208, 1936.
- 10 Mathias Herrmann and Alexander May. Attacking power generators using unravelled linearization: When do we output too much? In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 487–504, Tokyo, Japan, December 6–10 2009. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-10366-7\_29.
- 11 Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282, Shanghai, China, December 3–7 2006. Springer, Heidelberg, Germany. doi:10.1007/11935230\_18.
- 12 Simon Knellwolf and Willi Meier. Cryptanalysis of the knapsack generator. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 188–198, Lyngby, Denmark, February 13–16 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-21702-9\_11.
- 13 Donald Ervin Knuth. *The art of computer programming, Volume 4B: Combinatorial Algorithms, Part 2*. Addison-Wesley, 2022.
- 14 Florette Martinez. Attacks on pseudo random number generators hiding a linear structure. In Steven D. Galbraith, editor, *Topics in Cryptology – CT-RSA 2022*, volume 13161 of *Lecture Notes in Computer Science*, pages 145–168, Virtual Event, March 1–2 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-95312-6\_7.
- 15 S.M. Ross. *Probability Models for Computer Science*. Elsevier Science, 2002.
- 16 Rainer A. Rueppel and James L. Massey. Knapsack as a nonlinear function. In *IEEE International Symposium on Information Theory*. IEEE Press, NY, 1985.
- 17 Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT’97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15 1997. Springer, Heidelberg, Germany. doi:10.1007/3-540-69053-0\_18.
- 18 Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 93–110, Santa Barbara, CA, USA, August 18–22 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-45708-9\_7.
- 19 Joachim von zur Gathen and Igor Shparlinski. Predicting subset sum pseudorandom generators. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004: 11th Annual International Workshop on Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 241–251, Waterloo, Ontario, Canada, August 9–10 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-30564-4\_17.
- 20 Joachim von zur Gathen and Igor E. Shparlinski. Subset sum pseudorandom numbers: fast generation and distribution. *J. Math. Cryptol.*, 3(2):149–163, 2009. doi:10.1515/JMC.2009.007.
- 21 Lawrence C. Washington. *Elliptic curves. Number theory and cryptography*. Boca Raton, FL: Chapman and Hall/CRC, 2nd ed. edition, 2008.

**A Proof of Theorem 1**

We now proceed to prove theorem 1.

**Proof.** Let  $x, y, z, u, v$  denote five independent random bits, and set:

$$\begin{aligned} \rho &= \Pr(x + y = z) \\ \sigma &= \Pr(u + v = z \mid x + y = z) \\ \tau &= \Pr(u + y = v \mid x + y = z) \end{aligned}$$

We already know that  $\rho = 3/8$ . Building a simple table as above shows that  $\sigma = \tau = 5/12$  (see Table 7).

■ **Table 7** Tabulating  $u + v, x + y, u + y$  for  $x, y, z, u, v \in \{0, 1\}$ .

$u$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$v$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$x$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
$y$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
$z$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$u + v$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$x + y$	0	0	1	1	1	1	2	2	0	0	1	1	1	1	2
$u + y$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

$u$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$v$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$x$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
$y$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
$z$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$u + v$	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
$x + y$	0	0	1	1	1	1	2	2	0	0	1	1	1	1	2
$u + y$	1	1	2	2	1	1	2	2	1	1	2	2	1	1	2

Let  $X(i, j, k)$  denote the binary random variable that takes the value 1 if and only if  $A[i] + B[j] = C[k]$ , so that  $Y = \sum X(i, j, k)$ . Unless mentioned otherwise, all sums are taken over  $0 \leq i, j, k < N$ ; we omit the indices to alleviate notations.

The expected value of  $Y$  is easy to determine. Because the elements of the lists are identically distributed,  $\Pr(A[i] + B[j] = C[k])$  is independent of  $i, j$  and  $k$  and its value is  $\rho^n$ . We get:

$$E(Y) = E\left(\sum X(i, j, k)\right) = \sum E(X(i, j, k)) = \sum \Pr(A[i] + B[j] = C[k]) = N^3 \left(\frac{3}{8}\right)^n.$$

Because  $Y$  is the sum of binary random variables, we are entitled to use the “conditional expectation inequality” [15] (see also [13, MPR]):

$$\Pr(Y > 0) \geq \sum_{i=1}^n \frac{E(Y_i)}{E(Y \mid Y_i = 1)}. \tag{6}$$

Which, in our case, gives:

$$\Pr(Y > 0) \geq \sum \frac{E(X(i, j, k))}{E(Y \mid X(i, j, k) = 1)}.$$

As argued above, the value of the term under the sum is independent of  $i, j$  and  $k$ , so this boils down to:  $\Pr(Y > 0) \geq \left(\frac{3}{8}\right)^n / E(Y \mid X(0, 0, 0) = 1)$ . It remains to compute the expected number of good triplets under the assumption that there is at least one. This yields:

$$E(Y \mid X(0, 0, 0) = 1) = \sum \Pr(A[i] + B[j] = C[k] \mid A[0] + B[0] = C[0])$$

We split this sum into 8 parts by considering separately the situation where  $i = 0$ ,  $j = 0$  and  $k = 0$  (resp.  $\neq 0$  for each summation index). We introduce the shorthand  $p_{ijk} = \Pr(A[i] + B[j] = C[k] \mid A[0] + B[0] = C[0])$  and we assume that  $i, j, k > 0$ . Because  $A[i]$  is sampled independently from  $A[0]$  (resp.  $B, C$ ), the two events inside the conditional probability are in fact independent and therefore  $p_{ijk} = \left(\frac{3}{8}\right)^n$ . But when at least one index is zero, this is no longer the case. The extreme situation is  $p_{000} = 1$ .

When there is a single non-zero summation index, the situation is rather simple. If  $x + y = z$ , then  $x + U = z$  if and only if  $U = y$ , and this happens with probability  $2^{-n}$  because  $U$  is uniformly random. This shows that  $p_{i00} = p_{0j0} = p_{00k} = 2^{-n}$ .

It remains to deal with the case of two non-zero summation indices. In fact,  $p_{ij0}$  is simply  $\sigma^n$ , while both  $p_{i0k}$  and  $p_{0jk}$  are equal to  $\tau^n$  (by the symmetry between the role of the first two lists).

It follows that

$$\begin{aligned} E(Y \mid X(0, 0, 0) = 1) &= (N-1)^3 \left(\frac{3}{8}\right)^n + 3(N-1)^2 \left(\frac{5}{12}\right)^n + 3(N-1) \cdot 2^{-n} + 1 \\ &= N^3 \left(\frac{3}{8}\right)^n + 3N^2 \left(\frac{5}{12}\right)^n + 3N2^{-n} + 1 - \Delta \end{aligned}$$

$$\text{with } \Delta = (3N^2 - 3N + 1) \left(\frac{3}{8}\right)^n + 3(2N - 1) \left(\frac{5}{12}\right)^n + 3 \cdot 2^{-n}.$$

The “error term”  $\Delta$  is always positive for  $N \geq 1$ . Going back to the beginning, we have:

$$\begin{aligned} \Pr(Y > 0) &\geq \frac{N^3(3/8)^n}{N^3(3/8)^n + 3N^2(5/12)^n + 3N(1/2)^n + 1 - \Delta} \\ &\geq \frac{1}{1 + 3N^{-1}(10/9)^n + 3N^{-2}(4/3)^n + N^{-3}(8/3)^n} \end{aligned}$$

Using the convexity of  $x \mapsto 1/(1+x)$ , we obtain

$$\Pr(Y = 0) \leq 3N^{-1}(10/9)^n + 3N^{-2}(4/3)^n + N^{-3}(8/3)^n. \quad \blacktriangleleft$$



# The Compositional Structure of Bayesian Inference

**Dylan Braithwaite**

Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK

**Jules Hedges**

Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK

**Toby St Clere Smithe**

Topos Institute, Berkeley, CA, USA

Department of Experimental Psychology, University of Oxford, UK

---

## Abstract

Bayes' rule tells us how to invert a causal process in order to update our beliefs in light of new evidence. If the process is believed to have a complex compositional structure, we may observe that the inversion of the whole can be computed piecewise in terms of the component processes. We study the structure of this compositional rule, noting that it relates to the lens pattern in functional programming. Working in a suitably general axiomatic presentation of a category of Markov kernels, we see how we can think of Bayesian inversion as a particular instance of a state-dependent morphism in a fibred category. We discuss the compositional nature of this, formulated as a functor on the underlying category and explore how this can be used for a more type-driven approach to statistical inference.

**2012 ACM Subject Classification** Theory of computation → Categorical semantics; Mathematics of computing → Probabilistic representations; Mathematics of computing → Bayesian computation

**Keywords and phrases** monoidal categories, probabilistic programming, Bayesian inference

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.24

**Related Version** This paper contains material from unpublished preprints [16] and [3].

*Extended Version:* <https://arxiv.org/abs/2305.06112>

**Funding** *Dylan Braithwaite:* This author is funded through an NPL Industrial CASE Studentship.

## 1 Introduction

A *Markov kernel*  $f$  is a function whose output is stochastic given the input. It can be thought of as a stateless device which, if supplied with inputs, will produce an output that depends probabilistically on the input. Mathematically speaking, this is nothing but a conditional distribution  $\mathbb{P}(y|x)$ . The problem of Bayesian inversion is that we observe the output of such a device but only have a probabilistic “prior” belief about the input, and we would like to update our beliefs about what the input was given the output. For example, suppose a process takes the roll of an unseen die and tells us whether it was even or odd, except with probability 10% it flips the result. If the process tells us that a specific roll resulted in “even” then Bayes' law tells us how we should update a prior belief that the die is uniformly random, to obtain a belief about what the specific roll was.

It is commonly the case that the process is not a black box, but is a composite process formed from simpler pieces. The conditional distribution formed by sequentially composing two such stochastic functions is given by forming their joint distribution and then marginalising over the middle variable. In the case of a long chain of sequentially composed functions, belief updating for the whole can be done “compositionally” in terms of belief updating for each individual part, in a process notably similar to backpropagation in neural networks. A process such as this underlies probabilistic programming languages, which are able to run programs “backwards” after conditioning on some output.



© Dylan Braithwaite, Jules Hedges, and Toby St Clere Smithe;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 24; pp. 24:1–24:15

Leibniz International Proceedings in Informatics



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

24:2 The Compositional Structure of Bayesian Inference

The goal of this paper is to study this process of compositional Bayesian inversion of Markov kernels in isolation, using a suitable axiomatisation of a category of Markov kernels. We provide a method to build categories whose morphisms are pairs of a Markov kernel and an associated “Bayesian inverter”, which is itself built compositionally.

Symmetric monoidal categories with compatible families of copy and delete morphisms have been identified as an expressive language for synthetically representing concepts from probability theory [5, 8]. The typical interpretation given is that the objects represent sets or measurable spaces, and morphisms are Markov kernels between them. Branded *Markov categories* [8] in this context, these categories have recently seen widespread use in applied category theory as a foundation of probability for two reasons: they allow working axiomatically while abstracting over the specific details of theories such as stochastic matrices, Gaussian kernels and Giry (measure-theoretic) kernels; and they provide a rich string-diagram calculus allowing for simple graphical presentations of many calculations and constructions in probability.

$\mathbb{P}_X(x)\mathbb{P}_Y(y)$		$\sum_{y \in Y} \mathbb{P}_{X \times Y}(-, y)$	
Product distributions	Parallel composition	Marginalisation	Deletion
$\mathbb{P}_X(x)\delta(x, x')$		$\sum_{y \in Y} \mathbb{P}_f(z y)\mathbb{P}_g(y x)$	
Diagonal distributions	Copying	Chapman-Kolmogorov	Sequential composition

■ **Figure 1** The graphical representation of common formulas for probability distributions.

A categorical translation of Bayes’ law allows for a general definition of a *Bayesian inverse* to a morphism in a Markov category. However, in contrast to many other contexts where we have a notion of dualising morphisms, Bayesian inverses depend on an extra piece of data: the prior distribution. This is abstracted in our definition as a *state* on an object  $X$ , or a morphism out of the monoidal unit,  $I \rightarrow X$ , which represents a (non-conditional) probability distribution on  $X$ . This leads us to the abstract definition of a Bayesian inverse [5]:

► **Definition 1** (Bayesian Inversion). *Let  $f : X \rightarrow Y$  and  $p : I \rightarrow X$ , a kernel  $f' : Y \rightarrow X$  is called a Bayesian inverse of  $f$  at  $p$  if the following equation holds:*

This categorical based approach to Bayes’ law naturally leads to a question of whether Bayesian inverses compose; that is, can we construct a Bayesian inverse of a composite kernel  $f \circ g$  as a composite of the separate Bayesian inverses of  $f$  and  $g$ ? Inspecting the relevant equation

(1)

we can observe that this is indeed true if  $f'$  and  $g'$  are Bayesian inverse to  $f$  and  $g$ , but we have to ask for  $f'$  not to be inverse to  $f$  at  $p$ , but rather at  $g \circ p$ : in some sense the type of this composition is dependent on the kernels that it is being applied to. In other words, assuming a hypothetical function  $\text{BayesInv}(f, p)$  which computes Bayesian inverses, the composite could be expressed as

$$\text{BayesInv}(f \circ g, p) = \text{BayesInv}(g, p) \circ \text{BayesInv}(f, g \circ p)$$

Notably this has a similar form to the reverse-mode chain rule for Jacobian matrices that underlies backpropagation,  $J_{g \circ f}^\top(x) = J_g^\top(f(x))J_f^\top(x)$ . As such we think of the composition rule as a *chain rule for Bayesian updating*. To formalise the compositionality of such a construction we would typically like to promote it to a functor on our Markov category, which we denote in general as  $\mathcal{C}$ . The fact that Bayesian inverses are indexed by  $p$  in this way makes choosing the target of such a a Bayesian inversion functor nontrivial however.

One approach to making Bayesian inversion functorial, taken by Cho and Jacobs [5], is to work instead in a category where the objects  $X$  are equipped with a choice of  $p : I \rightarrow X$ : in this case, there is always a canonical choice of inverse morphism, making Bayesian inversion into a “dagger” functor. Although such a setting is still a Markov category, this approach does depart from the operational interpretation of morphisms as stochastic kernels, because the distribution produced is already present in the type of the morphism. So the morphisms have more of a “relational” role.

In this paper we take an alternative approach and instead consider a category where the morphisms are indexed families of kernels. Consequently, we can represent the entire  $(I \rightarrow X)$ -indexed family of Bayesian inverses as a single morphism, resolving the problems faced previously. The morphisms of this category have a very similar structure to a *lens* from database theory and functional programming [7], but, instead of performing deterministic updates to data structures, this performs Bayesian updates to beliefs. We therefore call such pairs *Bayesian lenses*.

This category is however “too big” in the sense that it contains everything whose type matches that of the Bayesian inverse. Rather than a failure of the abstraction, we view this as a feature allowing us to encode not only exact Bayesian inversion, but also approximate updaters and other structures. To pick out the specific lenses corresponding to the actual Bayesian inverse of  $f$  we therefore use a functor  $\mathcal{C} \rightarrow \mathbf{BLens}$ , which encodes a choice of Bayesian inverse for each kernel.

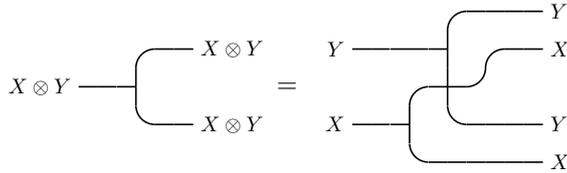
The category of Bayesian lenses is constructed as a fibred category that is closely related to the families fibration, commonly used in the semantics of dependent types. We embrace this relationship by extending  $\mathbf{BLens}$  into a larger category constructed from a generalised families fibration which leads us to a category of *dependent Bayesian lenses*, which have not only indexed families as morphisms, but indexed objects as well. This construction admits Bayesian inverses whose domains are allowed to depend on the prior distribution at which the inverse is taken. In certain Markov categories this allows us to consider inverses as being restricted to the support of the prior, which puts the construction on a neater theoretical foundation, and which clarifies thinking about the Bayesian inversion of structure maps in the category. Since Bayes’ law does not define how beliefs should be updated after a zero-probability observation, this falls into the usual pattern in computer science of using a type system to “make illegal states unrepresentable”<sup>1</sup>.

<sup>1</sup> This phrase originated with Yaron Minsky in the blog post <https://blog.janestreet.com/effective-ml-revisited/>

**2 Preliminaries**

► **Definition 2** (Markov Categories [8]). A Markov category is a symmetric monoidal category  $\mathcal{C}$  with the structure of a commutative comonoid on each object, such that:

- the assignment of counits (“delete” or “discard” maps) to objects is natural; and
- comultiplications (“copy” maps) are compatible with the monoidal structure:



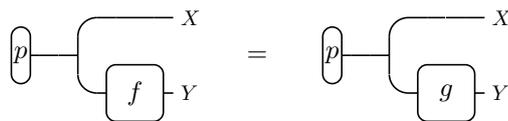
► **Example 3** (Stochastic Matrices, [8] example 2.5). There is a Markov category **FinStoch** whose objects are finite sets, and whose morphisms  $f : X \rightarrow Y$  are stochastic matrices  $f : X \times Y \rightarrow [0, 1]$ , i.e. satisfying  $\sum_{y \in Y} f(x, y) = 1$  for all  $x \in X$ . Identity morphisms are identity matrices, and composition of morphisms is by matrix multiplication, known in this context as the Chapman-Kolmogorov equation,  $(g \circ f)(x, z) = \sum_{y \in Y} f(x, y) \cdot g(y, z)$ . The symmetric monoidal structure of **FinStoch** is given on objects by cartesian product, and on morphisms by tensor product of stochastic matrices:  $(f \otimes g)((x, x'), (y, y')) = f(x, y) \cdot g(x', y')$ .

► **Example 4** (Gaussian Kernels, [8] section 6). There is a Markov category **Gauss** whose objects are Euclidean spaces  $\mathbb{R}^n$ , and whose morphisms  $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$  are triples of a matrix  $M \in \mathbb{R}^{m \times n}$ , a vector  $\mu \in \mathbb{R}^n$  and a positive semidefinite matrix  $\sigma \in \mathbb{R}^{n \times n}$ , considered to represent an affine function with independent Gaussian noise  $f(v) = Mv + \mathcal{N}(\mu, \sigma)$ . The definitions of categorical composition and monoidal product are slightly involved, but can be derived from laws of Gaussian probability.

► **Example 5** (Probability Kernels, [8] section 4). Giry [10] introduced a monad  $\mathcal{G}$  on the category of measurable spaces, now known as the Giry monad, taking each measurable space to the space of probability measures on it, equipped with an appropriate measurable structure. The Kleisli category  $\text{Kl}(\mathcal{G})$  is a Markov category, also known as **Stoch**, which allows working with arbitrary measure-theoretic probability and contains many other important Markov categories as subcategories, including the previous two examples. Although this category is in a sense canonical, it suffers from many undesirable properties due to extremely general nature of measure theory. One example which is relevant in the later sections is the spaces representing the support of certain distributions.

Keeping these examples in mind, we refer to the morphisms of such a category as *Markov kernels* or more simply as *kernels*.

► **Definition 6** (Almost-Sure Equality [5]). A pair of kernels  $f, g : X \rightarrow Y$  are  $p$ -almost-surely equal for some  $p : I \rightarrow X$ , if the following equality holds



In this case we write  $f \simeq_p g$ .

We note that this equation is similar in shape to that which defines a Bayesian inverse. Indeed this pattern of considering a kernel composed onto one branch of a copy is a standard way to consider a kernel in the context of a distribution on its domain. The appearance of this pattern in the definition of the Bayesian inverse means exactly that we are defining the concept only up to almost-sure equality.

► **Proposition 7** (Bayesian inverses are almost-surely equal). *For any kernel  $f : X \rightarrow Y$  and state  $p : I \rightarrow X$ , if two kernels  $g, g' : Y \rightarrow X$  both satisfy the conditions of a Bayesian inverse to  $f$ , then  $g \simeq_{f \circ p} g'$ .* ◀

Considering the interpretation of these equations, this result makes intuitive sense. It essentially says that Bayesian inversion is uniquely defined only on the points where Bayes' law would not have you divide by zero. This allows us to be mindful of the partiality of Bayes' law without requiring the added complication of considering partial maps. However, this will complicate matters later when we establish the functoriality of Bayes' law. In fact, as we will see, we will need some extra coherence assumptions to ensure that Bayesian inversion is functorial, rather than only almost-surely functorial.

### 3 Bayesian Updates Compose Optically

The Bayesian inverse of a kernel is defined with respect to a prior state on the domain of the kernel. We want to consider the general inverse of a kernel  $A \rightarrow B$  as a function which assigns to each prior  $p : I \rightarrow A$ , a  $p$ -inverse  $B \rightarrow A$ . However the space of priors with respect to which a kernel can be inverted depends on the domain of the kernel: it is the set of states  $\mathcal{C}(I, \text{dom}(f))$ . To formalise this dependence we construct an *indexed category*. Just as indexed sets  $(X_i)_{i \in I}$  can be thought of as functions  $I \rightarrow \mathbf{Set}$ , we define indexed categories as (pseudo)functors into  $\mathbf{Cat}$ . We consider Bayesian inverses in the context of an indexed category  $\mathcal{C} \rightarrow \mathbf{Cat}$  sending  $X$  to a category of  $\mathcal{C}(I, X)$ -indexed kernels.

► **Definition 8** (The  $\mathbf{Stat}$  construction). *We define the indexed category of  $\mathcal{C}$ -state-indexed kernels,  $\mathbf{Stat} : \mathcal{C}^{op} \rightarrow \mathbf{Cat}$ , as follows.*

For each  $X$ ,  $\mathbf{Stat}(X)$  is the category of  $\mathcal{C}(I, X)$ -indexed kernels, where

- objects are the objects of  $\mathcal{C}$ ;
- morphisms  $A \rightarrow B$  are functions  $\mathcal{C}(I, X) \rightarrow \mathcal{C}(A, B)$ ; and
- composition and identities are pointwise given by the corresponding structure in  $\mathcal{C}$ .

For  $f : X \rightarrow Y$  in  $\mathcal{C}$  we obtain a reindexing functor  $f^* : \mathbf{Stat}(Y) \rightarrow \mathbf{Stat}(X)$  which

- acts as the identity on objects; and
- reindexes functions by sending  $\sigma : \mathcal{C}(I, Y) \rightarrow \mathcal{C}(A, B)$  to  $f^* \sigma$  defined by:

$$\begin{array}{ccc} \mathcal{C}(I, X) & \longrightarrow & \mathcal{C}(I, Y) \longrightarrow \mathcal{C}(A, B) \\ p & \longmapsto & f \circ p \longmapsto \sigma(f \circ p) \end{array}$$

Since the reindexing functors are defined by pre-composition it is easy to verify that  $\mathbf{Stat}$  is indeed a functor.

This provides sufficient expressive power to represent general Bayesian inverses. For a kernel  $f : X \rightarrow Y$ , the general Bayesian inverse of  $f$  is a morphism  $Y \rightarrow X$  in  $\mathbf{Stat}(X)$ . However it is awkward to have to work across a large collection of categories in order to represent the inverses for every kernel. Really we would like to assemble the collection of

## 24:6 The Compositional Structure of Bayesian Inference

categories into a single category containing all of the inverses. Fortunately this category is described exactly by the *Grothendieck construction* [2, §8.3]. The Grothendieck construction realises an equivalence between indexed categories and *fibrations* – functors with a property of being suitably projection-like. The core of the construction is a disjoint union of the constituent categories, equipped with morphisms induced by the reindexing functors that connect each of the otherwise disjoint components.

$$\frac{\text{Indexed categories} \quad F : \mathcal{C} \rightarrow \mathbf{Cat}}{\text{Fibrations} \quad P_F : \left( \coprod_{C \in \mathcal{C}} F(C) \right) \rightarrow \mathcal{C}}$$

We proceed to construct the category of all state-indexed kernels in this way, which we will call the category of *Bayesian lenses*.

► **Definition 9** (Bayesian Lenses). *We define the category of Bayesian lenses as the Grothendieck construction  $\mathbf{BLens}(\mathcal{C}) = \coprod_{X \in \mathcal{C}} \mathbf{Stat}(X)^{op}$ .*

Explicitly this is a category whose objects are pairs  $\begin{pmatrix} X \\ A \end{pmatrix}$  of objects from  $\mathcal{C}$ , and whose morphisms are of the form  $(f, f^\#) : \begin{pmatrix} X \\ A \end{pmatrix} \rightarrow \begin{pmatrix} Y \\ B \end{pmatrix}$  where  $f$  is a kernel  $X \rightarrow Y$ , and  $f^\#$  is a family of “backward” kernels,  $\mathcal{C}(I, X) \rightarrow \mathcal{C}(B, A)$ . The composition of two such morphisms

$$\begin{pmatrix} X \\ A \end{pmatrix} \xrightarrow{(f, f^\#)} \begin{pmatrix} Y \\ B \end{pmatrix} \xrightarrow{(g, g^\#)} \begin{pmatrix} Z \\ C \end{pmatrix}$$

is  $(g \circ f, (g \circ f)^\#)$  where  $(g \circ f)^\#$  is a function  $\mathcal{C}(I, X) \rightarrow \mathcal{C}(C, A)$ , sending  $p : I \rightarrow X$  to the composition

$$C \xrightarrow{g^\#(f \circ p)} B \xrightarrow{f^\#(p)} A.$$

We call the morphisms of this category *Bayesian lenses*.

► **Remark 10.** Contrary to our general description of the Grothendieck construction, we in fact used the “fibrewise” opposite of  $\mathbf{Stat}$ , taking  $\mathbf{Stat}(X)^{op}$  for each  $X$  in  $\mathcal{C}$ . Ultimately this does not change the space of functions we can represent, but it provides a neater type signature for the morphisms we care about. Namely this means the Bayesian inverse of a kernel will be a morphism  $\begin{pmatrix} X \\ X \end{pmatrix} \rightarrow \begin{pmatrix} Y \\ Y \end{pmatrix}$ , rather than  $\begin{pmatrix} X \\ Y \end{pmatrix} \rightarrow \begin{pmatrix} Y \\ X \end{pmatrix}$ . We think of the appearance of the opposite here as signalling that we are working with a kind of bidirectional process.

Spivak [19] proposes that the fibrewise opposite of a fibration be considered as a generalisation of the lens construction from database theory and functional programming [7]. More clearly, if  $\mathcal{C}$  is the category of sets (which is degenerately a Markov category), then  $f^\# : \mathcal{C}(I, X) \rightarrow \mathcal{C}(B, A)$  can be equivalently written  $f^\# : X \times B \rightarrow A$ , and we recover the standard definition of a lens. This motivates our use of the term “Bayesian lens” for these morphisms.

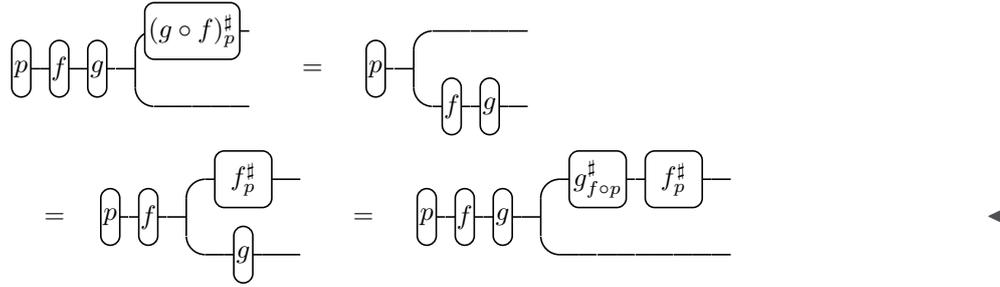
► **Proposition 11** (Bayesian inversion is almost functorial). *If  $\mathcal{C}$  has Bayesian inverses for ever kernel at every prior, then Bayesian inversion defines a functor  $T : \mathcal{C} \rightarrow \mathbf{BLens}(\mathcal{C})$  up to almost-sure equality.*

**Proof.** Because Bayesian inverses are only unique up to almost-equality,  $T$  is only almost surely functorial: it maps each object  $X$  to  $\begin{pmatrix} X \\ X \end{pmatrix}$ , and each kernel  $f : X \rightarrow Y$  to a lens  $(f, f^\#) : \begin{pmatrix} X \\ X \end{pmatrix} \rightarrow \begin{pmatrix} Y \\ Y \end{pmatrix}$  whose inverse component is given by a representative of the almost-surely unique family of Bayesian inversions of  $f$ .

Given a pair of composable kernels  $X \xrightarrow{f} Y \xrightarrow{g} Z$ , we need to check that this mapping is almost surely functorial. Following equation (1), we note that the state with respect to which this functoriality is almost sure is  $g \circ f \circ p$ , for every state  $p$  on  $X$ . We therefore need to verify that

$$\begin{pmatrix} X \\ X \end{pmatrix} \xrightarrow{(f, f^\#)} \begin{pmatrix} Y \\ Y \end{pmatrix} \xrightarrow{(g, g^\#)} \begin{pmatrix} Z \\ Z \end{pmatrix} \simeq_{g \circ f} \begin{pmatrix} X \\ X \end{pmatrix} \xrightarrow{(g \circ f, (g \circ f)^\#)} \begin{pmatrix} Z \\ Z \end{pmatrix}.$$

This follows from three applications of Definition 1:



Note that although Markov categories in general may not admit all Bayesian inverses, we can always restrict to a wide subcategory  $\mathcal{C}^\dagger$  consisting of only those kernels which admit inverses.

#### 4 Dependent Bayesian Lenses

Those familiar with categorical semantics of type theory might observe that the construction of Bayesian lenses is similar to that of the *families fibration* [12, §1.2]. This construction, commonly used in the interpretation of dependent types, constructs a category whose objects and morphisms are set-indexed families of objects and morphisms from another underlying category. As a fibration this category projects onto the category of sets, by picking out the indexing set or reindexing function under each family.

► **Definition 12** (The families fibration [12, §1.2]). *The indexed category of families over a category  $\mathcal{C}$  is the functor  $\mathbf{Fam}_{\mathcal{C}} : \mathbf{Set}^{op} \rightarrow \mathbf{Cat}$  defined as so:*

- For a set  $X$ ,  $\mathbf{Fam}_{\mathcal{C}}(X)$  is the category whose objects are  $X$ -indexed families of objects  $X \rightarrow \text{Ob}(\mathcal{C})$  and whose morphisms  $A \rightarrow B$  are families of morphisms  $\phi : \{x \in X\} \rightarrow \mathcal{C}(A_x, B_x)$ .
- For a function  $\alpha : Y \rightarrow X$ , the induced reindexing functor  $\alpha^* : \mathbf{Fam}_{\mathcal{C}}(X) \rightarrow \mathbf{Fam}_{\mathcal{C}}(Y)$  is given by pre-composition with alpha:

$$\alpha^*(A) : X \xrightarrow{\alpha} Y \xrightarrow{A} \text{Ob}(\mathcal{C})$$

$$\alpha^*(\phi) : \{y \in Y\} \xrightarrow{\alpha} \{\alpha(x) \in X\} \xrightarrow{\phi} \mathcal{C}(A_{\alpha(y)}, B_{\alpha(y)})$$

Alternatively, viewing sets as discrete categories,  $\mathbf{Fam}_{\mathcal{C}}$  is the functor that sends  $X$  to the functor category  $\mathbf{Cat}(X, \mathcal{C})$  and reindexes by precomposition with the function viewed as a functor between discrete categories.

A crucial difference between this and the indexed category in Definition 8 is that the latter is indexed only by sets of the form  $\mathcal{C}(I, X)$  whereas  $\mathbf{Fam}_{\mathcal{C}}(-)$  allows for arbitrary indexing sets. Having the indexing sets take the form  $\mathcal{C}(I, -)$  allows us to additionally restrict the reindexing functors to those represented by kernels of  $\mathcal{C}$ . Formally we can see this as the result of reindexing  $\mathbf{Fam}_{\mathcal{C}}(-)$  itself with the state functor:

## 24:8 The Compositional Structure of Bayesian Inference

► **Definition 13** (State-indexed families). *The indexed category  $\mathbf{StFam}_{\mathcal{C}} : \mathcal{C}^{op} \rightarrow \mathbf{Cat}$  is given by the composite of functors:*

$$\mathcal{C}^{op} \xrightarrow{\mathcal{C}(I, -)^{op}} \mathbf{Set}^{op} \xrightarrow{\mathbf{Fam}_{\mathcal{C}}} \mathbf{Cat}$$

$$\begin{array}{ccc} A & \mathcal{C}(I, A) & \mathcal{C}^{\mathcal{C}(I, A)} \\ f \downarrow & \downarrow f \circ - & \uparrow \alpha \mapsto \alpha(f \circ -) \\ B & \mathcal{C}(I, B) & \mathcal{C}^{\mathcal{C}(I, B)} \end{array}$$

*This has as objects  $\mathcal{C}(I, X)$ -indexed families of objects from  $\mathcal{C}$  and as morphisms  $\mathcal{C}(I, X)$ -indexed families of kernels from  $\mathcal{C}$ .*

Note that the categories indexed by  $\mathbf{StFam}$  are more general than those indexed by  $\mathbf{Stat}$ : the objects of  $\mathbf{StFam}_{\mathcal{C}}(X)$  are whole functions  $\mathcal{C}(I, X) \rightarrow \mathbf{Ob}(\mathcal{C})$ , whereas an object of  $\mathbf{Stat}_{\mathcal{C}}(X)$  is merely a single object of  $\mathcal{C}$ .

► **Proposition 14.**  *$\mathbf{Stat}_{\mathcal{C}}$  is a subfunctor of  $\mathbf{StFam}_{\mathcal{C}}$  given by restricting to subcategories which consist only of constantly-indexed families of objects i.e. those families  $\mathcal{C}(I, X) \rightarrow \mathbf{Ob}(\mathcal{C})$  which are constant functions.* ◀

Observing that  $\mathbf{StFam}$  is a generalisation of  $\mathbf{Stat}$ , we are led to consider whether the fibration constructed from  $\mathbf{StFam}$  could profitably be considered as a generalised category of lenses. By analogy with the use of  $\mathbf{Fam}$  in dependent type theory we refer to these as *dependent Bayesian lenses*.

► **Definition 15** (Dependent Bayesian Lenses). *We define the category of dependent Bayesian lenses over  $\mathcal{C}$  to be the Grothendieck construction of the fibrewise opposite of  $\mathbf{StFam}$ ,*

$$\mathbf{DBLens}(\mathcal{C}) = \coprod_{X \in \mathcal{C}} \mathbf{StFam}_{\mathcal{C}}(X)^{op}.$$

Unpacking this definition we have that:

- Objects of  $\mathbf{DBLens}(\mathcal{C})$  are pairs  $\binom{X}{A}$  where  $X \in \mathcal{C}$  and  $A : \mathcal{C}(I, X) \rightarrow \mathbf{Ob}(\mathcal{C})$ . We think of this as representing a set of dependent pairs, whose elements are  $\langle p, a \rangle \in \mathcal{C}(I, X) \times A(p)$ .
- Morphisms  $\binom{X}{A} \rightarrow \binom{Y}{B}$  are pairs  $(f, f^{\#})$  of a kernel  $f : X \rightarrow Y$  and a family of kernels  $f^{\#} : \{p \in \mathcal{C}(I, X)\} \rightarrow \mathcal{C}(B(f \circ p), A(p))$ .
- The composition is as with the non-dependent version in Definition 9. Aside from the fact that  $f^{\#}$  is here considered as a dependent function, the data of a morphism is the same. It is straightforward to verify that the extra dependent typing data still aligns where appropriate.

► **Remark 16.** The functor  $\mathcal{C}(I, -)^{op} : \mathcal{C}^{op} \rightarrow \mathbf{Set}^{op}$  can be made into a lax monoidal functor with structure maps induced by functions  $\mathcal{C}(I, X \otimes Y) \rightarrow \mathcal{C}(I, X) \times \mathcal{C}(I, Y)$  which map states to pairs of states obtained by deleting either variable. Then it follows by results in [14] that  $\mathbf{StFam}$  is the composition of lax monoidal functors, and so its Grothendieck construction inherits a monoidal structure.

### 5 Support Objects

Giving an abstract account of Bayes' law, Definition 1 promises to be very important in studying Bayesian statistics synthetically, but it is in some way unsatisfying because it only specifies a morphism up to almost-sure equality. For example, considering  $\mathbf{FinStoch}$ , if a

distribution  $p : I \rightarrow X$  is not fully supported, then the inverse of a kernel  $X \rightarrow Y$  is only specified by the above definition at the points in the support of  $p$ . We can work around this ambiguity however by instead considering inverses as kernels between objects representing the supports of distributions.

Fritz [8] proposes a definition for support objects in a Markov category, but does not develop the idea further. Here we investigate some properties of support objects and explain how they can be used to enhance the theory of abstract Bayesian inversion.

► **Definition 17.** Fix a state  $p : I \rightarrow X$ . An object  $X_p$  is called a support of  $p$  if  $X_p$  represents the covariant functor  $(\mathcal{C}(X, -)/\simeq_p) : \mathcal{C} \rightarrow \mathbf{Set}$ .

This definition succinctly captures the essential properties of the support of a distribution, but it is quite opaque and does not encourage intuition. If an object is a support of another object, intuitively it should behave as a subspace, having an inclusion map which satisfies some extra properties. We state this formally in terms of the existence of certain section-retraction kernels representing this inclusion.

► **Proposition 18.**  $X_p$  is a support of  $p : I \rightarrow X$  if and only if there is a section-retraction pair  $X_p \xrightarrow{i} X \xrightarrow{r} X_p$  such that for any kernels  $f, g : X \rightarrow Y$  we have  $f \simeq_p g \iff f \circ i = g \circ i$ .

**Proof.** Assume  $X_p$  is a support. This means we have a natural isomorphism  $\Phi : (\mathcal{C}(X, -)/\simeq_p) \rightarrow \mathcal{C}(X_p, -)$ . We take  $i = \Phi_X(\text{id}_X)$ , then we see from the following naturality square that the action of  $\Phi$  must be to pre-compose representative morphisms with  $i$ :

$$\begin{array}{ccc}
 i & \longleftarrow & [\text{id}_X]_{\simeq_p} \\
 \downarrow & & \downarrow \\
 \mathcal{C}(X_p, X) & \xleftarrow{\Phi_X} & \mathcal{C}(X, X)/\simeq_p \\
 \mathcal{C}(X_p, f) \downarrow & & \downarrow \mathcal{C}(X, f)/\simeq_p \\
 \mathcal{C}(X_p, Y) & \xleftarrow{\Phi_Y} & \mathcal{C}(X, Y)/\simeq_p \\
 \downarrow & & \downarrow \\
 f \circ i & \longleftarrow & [f]_{\simeq_p}
 \end{array}$$

Hence we establish the property that pre-composition by  $i$  is an isomorphism between kernels from  $X$  and  $\simeq_p$ -equivalences classes of kernels from  $X_p$ . We further have that  $\text{id}_{X_p} = \Phi(\Phi^{-1}(\text{id}_{X_p})) = \Phi^{-1}(\text{id}_X) \circ i$ , so we can take the retract to be  $r = \Phi^{-1}(\text{id})$ .

Conversely, given such an  $i$  and  $r$ , it is clear that pre-composition by  $i$  defines a function  $\mathcal{C}(X, Y) \rightarrow \mathcal{C}(X_p, Y)$  natural in  $Y$  and the assumed property of  $i$  guarantees that this is a bijection from  $\simeq_p$ -equivalence classes. Finally we have that  $i \circ r \circ i = i$ , so  $i \circ r \simeq_p \text{id}_{X_p}$ . Hence pre-composition by  $r$  is an inverse to  $(-)\circ i : (\mathcal{C}(X, -)/\simeq_p) \rightarrow \mathcal{C}(X_p, -)$ . ◀

While support objects are not necessarily unique, they must be unique up to isomorphism, since two support objects for the same distribution must by definition represent the same presheaf. When we discuss a given support object we therefore really mean a given support object along with a choice of section and retraction.

Now, if we have a distribution on  $X$ ,  $p : I \rightarrow X$ , and a kernel  $f : X \rightarrow Y$  we can push  $p$  forward to a distribution  $f \circ p$  on  $Y$ . So  $f$  restricts to a kernel  $f|_p = r \circ f \circ i : X_p \rightarrow Y_{f \circ p}$ . Dually we have an inclusion of kernels  $g : X_p \rightarrow Y_q$  into  $\langle g \rangle = i \circ g \circ r : X \rightarrow Y$ . Using these there is an obvious adjustment to the definition of Bayesian inversion in order to capture Bayesian inverses between supports:

## 24:10 The Compositional Structure of Bayesian Inference

► **Definition 19** (Bayesian-inverse-with-support). *Fix kernels  $p : I \rightarrow X$  and  $f : X \rightarrow Y$  with support objects  $X_p$  and  $Y_{f \circ p}$ . We call a kernel  $f_p^\# : Y_{f \circ p} \rightarrow X_p$  a Bayesian inverse with support if  $\langle f_p^\# \rangle$  is an ordinary Bayesian inverse for  $f$  at  $p$ .*

This is very similar to the previous definition. However the presence of support objects in  $f_p^\#$ 's domain and codomain mean that the kernel is more canonical than an ordinary Bayesian inverse. In fact for any ordinary inverse  $f_p^\#$ , its restriction to the distribution  $f \circ p$  is an inverse-with-support.

► **Theorem 20** (Unique Bayesian Inversion). *Fix kernels  $f : X \rightarrow Y$  and  $p : I \rightarrow X$ , and support objects  $X_p$  and  $Y_{f \circ p}$ . If  $f$  has an ordinary Bayesian inverse at  $p$ , then there is a unique inverse-with-support to  $f$  at  $p$ .*

**Proof.** See Appendix A. ◀

This final result suggests that we can now define a functor that picks out the canonical inverse for a given kernel. Recall that in Proposition 11 we were almost able to show that Bayesian inversion defines a functor  $\mathcal{C} \rightarrow \mathbf{BLens}(\mathcal{C})$  except that the functoriality constraint only holds up to almost-equality. Using support objects we improve this into strict functoriality between kernels. Bayesian inversion with supports departs from the previous situation slightly in that the domain and codomain of the inverse kernels vary as the indexing distribution varies. Fortunately this is exactly the extra level of generality afforded us by dependent Bayesian lenses.

► **Proposition 21.** *If  $\mathcal{C}$  has Bayesian inverses for every kernel, and support objects for every distribution  $I \rightarrow X$  then the fibred category  $\mathbf{DBLens}(\mathcal{C})$  has a section  $T : \mathcal{C} \rightarrow \mathbf{DBLens}(\mathcal{C})$  sending kernels to families of their Bayesian inverses between support objects.*

**Proof.** We write explicitly the mapping defining  $T$ . On objects  $T(X) = \begin{pmatrix} X \\ S_X \end{pmatrix}$  where  $S_X : \mathcal{C}(I, X) \rightarrow \text{Ob}(\mathcal{C})$  maps  $p$  to a choice of support object  $X_p$ . On kernels  $f : X \rightarrow Y$ , we have  $T(X) = (f, f^\#) : \begin{pmatrix} X \\ S_X \end{pmatrix} \rightarrow \begin{pmatrix} Y \\ S_Y \end{pmatrix}$  where for each  $p \in \mathcal{C}(I, X)$ ,  $f_p^\# : Y_{f \circ p} \rightarrow X_p$  is given by the Bayesian inverse with support of  $f$  at  $p$ .

The functoriality of  $T$  follows similarly to in Proposition 11. Briefly, if  $f$  and  $g$  are composable Bayesian inverses with support, then  $\langle f \rangle$  and  $\langle g \rangle$  are composable ordinary inverses, so by Proposition 11 their composition must be an inverse. By the definition of the inclusions it is immediately that  $\langle f \rangle \circ \langle g \rangle = \langle f \circ g \rangle$ , so  $f \circ g$  is an inverse-with-support. Then by uniqueness, the fact that the composition  $f \circ g$  is a Bayesian inverse implies the strict equality required for functoriality. ◀

► **Remark 22.** Proposition 21 proves the compatibility of sequential composition in a Markov category with Bayesian inversion, however there is another axis for composition in our graphical notation. This is the parallel composition, or monoidal product of  $\mathcal{C}$ . To ask for this structure to be compatible with inversion is to ask for the functor  $T$  to respect the monoidal structure. Indeed as we noted in Remark 16, we may naturally define a monoidal product on  $\mathbf{DBLens}(\mathcal{C})$ , then  $T$  straightforwardly inherits the structure of a *lax monoidal functor* with respect to this.

► **Remark 23.** Most of the results developed in this section relied on the existence of support objects. Indeed this is satisfied for our first two examples Example 3 and Example 4, but this is quite a strong assumption in general. The approach to functorial Bayesian inversion taken in [5] and later expanded on in [8] is to work in a category  $\mathbf{ProbStoch}(\mathcal{C})$  whose objects are equipped with a choice of distribution and whose morphisms are quotiented by

almost-sure equality with respect to these chosen distributions. This makes exact functoriality straightforward to prove, but it detracts from the interpretation of morphisms as stochastic functions. Working in  $\mathbf{ProbStoch}(\mathcal{C})$  we are unable to think of Bayesian updating as a dynamic process which maps observations to new distributions, since the resultant distribution is already encoded in the codomain.

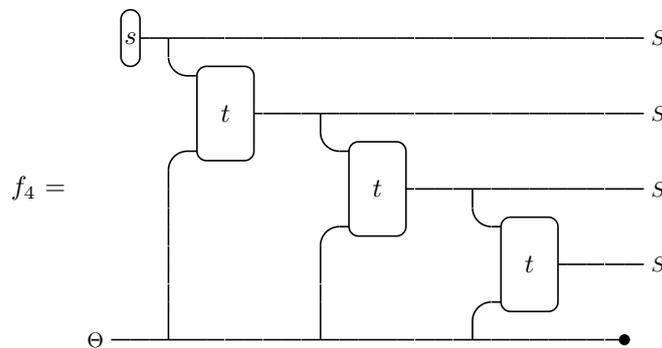
We could instead take a hybrid approach, using dependent Bayesian lenses where the forwards kernel is still valued in  $\mathcal{C}$  but the “backwards” mapping is in  $\mathbf{ProbStoch}(\mathcal{C})$ . This means the type of a general inverse can become a function  $\{p \in \mathcal{C}(I, X)\} \rightarrow \mathbf{ProbStoch}(\mathcal{C})((Y, f \circ p), (X, p))$ . This provides a lot of the same theoretical niceties of support objects, including exact functoriality, while still applying in a more general context. We leave the further investigation of this construction to future work.

## 6 Example: Estimating Transition Probabilities in Markov Chains

To illustrate the compositional nature of Bayesian inversion in practice we consider how to view a typical Bayesian inference problem in this framework. Namely we use as our example, the common problem of learning transition probabilities for a Markov chain from an observed sequence of states [21].

In the category  $\mathbf{FinStoch}$ , a Markov chain is nothing but an endomorphism  $t : S \rightarrow S$  for some finite set  $S$  of states, with an initial state distribution  $s : I \rightarrow S$ . Often we do not know the actual transition probabilities, but instead have a family of distributions dependent on another external parameter,  $t : S \otimes \Theta \rightarrow S$ . In such a situation we want to choose an optimal value of  $\Theta$  based on a sequence of observations of states in the past. A common technique for choosing this value is to consider the observed state sequence as an element of the space  $S \otimes \dots \otimes S$ , with which we can perform a Bayesian update. That is, given a state sequence  $(s_1, \dots, s_n)$  we would like to compute a posterior distribution  $\mathbb{P}(\Theta | s_1, \dots, s_n)$ .

From the data involved we can define a family of maps  $f_n : \Theta \rightarrow S^{\otimes n}$  describing the distribution over  $n$ -length state sequences corresponding to any value of  $\Theta$  (see Figure 2 for an example). Computing the Bayesian inverse of  $f_n$  we obtain kernels which map sequences  $S^{\otimes n}$  to posterior distributions over  $\Theta$ . Of course, as we have already discussed, in order to compute this all we need is the Bayesian inverse for the transition matrix  $t$ . Knowing the functorial relationship between “forward” models  $f_n$  and their corresponding inverse models, we can build an inverse for  $f_n$  out of many copies of the inverse for  $t$ .



■ **Figure 2** An example of a “state-trace” kernel obtained from a Markov chain, yielding state sequences of length 4.

## 24:12 The Compositional Structure of Bayesian Inference

Figure 2 depicts a typical composite which we may want to invert. It consists only of four kinds of non-trivial cell: the transition matrix  $t$ , the initial state distribution  $s$ , copy maps  $\text{copy}_S : S \rightarrow S \otimes S$  and  $\text{copy}_\Theta : \Theta \rightarrow \Theta \otimes \Theta$ , and a delete map  $\text{delete}_\Theta : \Theta \rightarrow I$ . As  $t$  is an arbitrary kernel there is little we can say about its Bayesian inverse, but we can fully characterise the inverses for the other kinds of cells. The inverse of  $s$  at  $p$  should have a type  $s_p^\# : S_p \rightarrow I$ , but by the uniqueness of the delete map this is equal to  $\text{delete}_{S_p} : S_p \rightarrow I$ . Dually, filling in the definition of Bayesian inversion for  $\text{delete}_\Theta$  at  $p$  requires that  $(\text{delete}_\Theta)_p^\#$  is exactly  $p : I \rightarrow \Theta$ .

Finally we consider the Bayesian inverse of  $\text{copy}_S$ . Of course the case for  $\text{copy}_\Theta$  is identical. In the case of non-dependent Bayesian lenses, without supports, it is difficult to understand what these copy maps do. The copy operation is supported on only a small subset of its domain. Intuitively this is the diagonal set  $\{(s, s) | s \in S\}$ , but in the abstract case this set notation is not valid. We may wonder if there is a way that we can describe this using only the axiomatic setting we have been working with so far. Indeed, we can show that there is in fact an isomorphism of the support object  $(S \otimes S)_{\text{copy}_\Theta \circ p} \cong S_p$  and moreover, the morphism witnessing this isomorphism is exactly the Bayesian inverse of  $\text{copy}$ .

This isomorphism is a very powerful fact. It enforces in the type of a morphism, using the notion of dependent typing present in  $\mathbf{DBLens}(\mathcal{C})$ , that the observations we make from a process involving copying should preserve the exact equalities expected given our knowledge about the generative processes from which the observations originated.

It is not difficult to derive an explicit formula for this with traditional probability theory, but our structural viewpoint can offer a more pedagogical approach to describing inference algorithms. Using a 2-dimensional syntax for kernels and understanding the compositional nature of Bayesian inversion, a lot of the apparent complexity of equations in statistics is clarified by applying operations piecewise on string diagrams. It additionally becomes straightforward to see how to extend this to more complicated scenarios. For example, a hidden Markov model is obtained from the above example simply by postcomposing each wire with an additional kernel  $o : S \rightarrow O$ . Then compositionality described exactly how to extend a solution to account for the additional mapping.

This approach provides not just additional pedagogy, but also is very amenable to algorithmic study: as future work, we hope to investigate ways in which, by isolating the basic repeated units and their compositional structure, we can automatically generate a lot of the additional data required and can perhaps facilitate certain optimisations.

### 7 Further work

Our work situates Bayesian inversion amongst a range of bidirectional processes observed in different contexts, each of which exhibit lens (or lens-like) structure and “cybernetic” application [4]: reverse-mode automatic differentiation (generalizing backpropagation of error) [6]; economic games [9]; reinforcement learning [11]; and database updating [7]. It was in this latter setting, in the context of functional programming, that lenses were first described, and generalizations of lenses remain in popular use in that setting. This points to a first avenue for future work: a new approach to probabilistic programming that incorporates both Bayesian and differential updating, extending the currently popular use of deterministic lenses, and with general application to cybernetic systems.

Probabilistic programming languages allow the programmer essentially to construct Markov kernels in a compositional way using a standard programming language syntax, and perform inference on the resulting models [20]. Typically, performing inference is not

transparently compositional, and so we hope that our results will lead to improvements here, for instance by allowing the programmer to “amortize” parts of an inference process. Perhaps more importantly, our approach could lead to the use of a dependent type system to statically rule out events of measure zero. Furthermore, in a generically bidirectional language of the kind we envisage, we expect that compilers will be able to share optimizations across differential and probabilistic parts.

We expect these ideas not only to be of use to language designers therefore, but also to users. In many applications of probabilistic inference, one first constructs a “joint model”: a joint state across all the variables of interest, which may factorise according to a Bayesian network or other graphical model. Because computing exact Bayesian inversions involves marginalization (normalization: the sum or integral in the Chapman-Kolmogorov equation), and such a computation is often intractable, one usually resorts to approximate methods, and this is where the “extra” morphisms in categories of Bayesian lenses come in: they can be understood as approximate inverses.

By parameterizing these lenses, one can associate to them *loss functions* that characterize “how far” a given inversion is from optimality<sup>2</sup>; like the inversions themselves, these losses depend on both the priors and the observations, and they again compose according to a lens pattern. The third author, in the unpublished works [17, 18], has begun developing this programme, based on ideas from compositional game theory [9]. It results in an account of approximate inference whose algorithms are “correct by construction”, which may be an advantage over traditional methods which simply start from a given joint distribution. Moreover, because the loss functions are “local” to each lens, the resulting framework captures the compositional structure seemingly exhibited by “predictive coding” neural circuits in the brain [1]. Similarly, by making use of the lens structure presented here, the framework suggests a formal unification of backprop and predictive coding, as sought by various authors [13, 15], and it reveals connections to the method of backward induction in reinforcement learning [11]. We hope that future developments make use of these relationships, so that we may build intelligent systems that are both efficient and well-understood.

---

## References

- 1 Andre M. Bastos, W. Martin Usrey, Rick A. Adams, George R. Mangun, Pascal Fries, and Karl J. Friston. Canonical microcircuits for predictive coding. *Neuron*, 2012. doi:10.1016/j.neuron.2012.10.038.
- 2 Francis Borceux. *Handbook of Categorical Algebra 2. Categories and Structures*. Cambridge University Press, 1994. doi:10.1017/CB09780511525865.
- 3 Dylan Braithwaite and Jules Hedges. Dependent Bayesian Lenses: Categories of Bidirectional Markov Kernels with Canonical Bayesian Inversion, 2022. arXiv:2209.14728.
- 4 Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Towards Foundations of Categorical Cybernetics. *ACT21*, 2022. doi:10.4204/EPTCS.372.17.
- 5 Kenta Cho and Bart Jacobs. Disintegration and Bayesian inversion via string diagrams. *MSCS*, 2019. doi:10.1017/S0960129518000488.
- 6 Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. *ESOP22*, 2022. doi:10.1007/978-3-030-99336-8\_1.

---

<sup>2</sup> Examples of such loss functions include the relative entropy (Kullback-Leibler divergence) and variational free energy (“evidence lower bound”).

- 7 J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *TOPLAS*, 2007. doi:10.1145/1232420.1232424.
- 8 Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 2020. doi:10.1016/j.aim.2020.107239.
- 9 Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. *LICS18*, 2018. doi:10.1145/3209108.3209165.
- 10 Michèle Giry. A categorical approach to probability theory. *Categorical aspects of Topology and Analysis*, 1982. doi:10.1007/BFb0092872.
- 11 Jules Hedges and Riu Rodríguez Sakamoto. Value Iteration Is Optic Composition. *ACT22*, 2023. Forthcoming.
- 12 Bart Jacobs. *Categorical Logic and Type Theory*. Elsevier Science, 1999.
- 13 Beren Millidge, Tommaso Salvatori, Yuhang Song, Rafal Bogacz, and Thomas Lukasiewicz. Predictive coding: Towards a future of deep learning beyond backpropagation? *IJCAI22*, 2022. doi:10.24963/ijcai.2022/774.
- 14 Joe Moeller and Christina Vasilakopoulou. Monoidal Grothendieck construction, 2020.
- 15 Robert Rosenbaum. On the relationship between predictive coding and backpropagation. *PLoS ONE*, 2022. doi:10.1371/journal.pone.0266102.
- 16 Toby St. Clere Smithe. Bayesian updates compose optically, 2020. arXiv:2006.01631.
- 17 Toby St. Clere Smithe. Compositional Active Inference I: Bayesian Lenses. *Statistical Games*, 2022. arXiv:2109.04461.
- 18 Toby St Clere Smithe. *Mathematical Foundations for a Compositional Account of the Bayesian Brain*. PhD thesis, University of Oxford, 2023. arXiv:2212.12538.
- 19 David I. Spivak. Generalized Lens Categories via functors  $\mathcal{C}^{\text{op}} \rightarrow \text{Cat}$ , 2019. arXiv:2109.04461.
- 20 Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming, 2021. arXiv:1809.10756.
- 21 H. Wu and F. Noé. Maximum a posteriori estimation for markov chains based on gaussian markov random fields. *ICCS10*, 2010. doi:10.1016/j.procs.2010.04.186.

## A Uniqueness of Bayesian Inversion

► **Notation 24.** *In the graphical language we denote the inclusion and restriction kernels for a support object respectively as follows:  $X_p \text{ } \leftarrow \vdash \text{ } X \text{ } \text{ } \vdash \text{ } X_p$*

This is a convenient shorthand for indicating the section-retract relation between sections and restrictions, however care must be taken when calculating with these. Since we are not labelling the triangles, we should only pair inclusions and restrictions corresponding to the same states; the characterising equations only hold for such pairs. However in practice it is difficult to naturally arrive at a situation where this caveat can be an issue, so this is less of a shortcoming that it might seem.

We have strict equality of inclusions followed by restrictions, but in the converse we have almost-equality of restrictions followed by inclusions.

► **Lemma 25.** *Restriction is almost inverse to inclusion:*

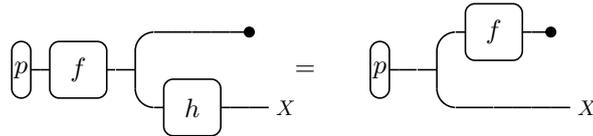
$$X \xrightarrow{r} X_p \xrightarrow{i} X \simeq_p X \xrightarrow{id} X.$$

**Proof.** Note that  $i \circ r \circ i = i$  so by the quotienting property of  $(-)\circ i$  we have  $i \circ r \simeq_p \text{id}$ . ◀

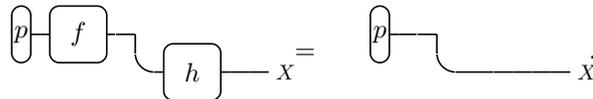
► **Proposition 26.** *Fix kernels  $f : X \rightarrow Y$  and  $p : I \rightarrow X$ , and support objects  $X_p$  and  $Y_{fp}$ . Then inverses-with-support of  $f$  at  $p$  are in bijection with  $\simeq_{fp}$ -equivalence classes of ordinary Bayesian inverses.*

**Proof.** We first exhibit a map  $\Psi$  from inverses-with-support to ordinary inverses. Let  $g : Y_{fp} \rightarrow X_p$  be a Bayesian inverse with support of  $f$  at  $p$ . By the definition of inverses with support this means that  $\Psi(g) := \langle g \rangle$  is an ordinary Bayesian inverse.

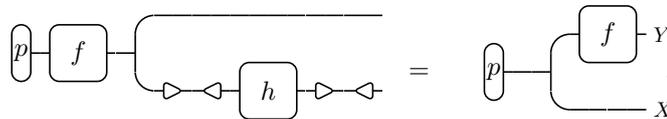
Conversely if  $h : Y \rightarrow X$  is an ordinary Bayesian inverse to  $f$  at  $p$ , we want to define an inverse with support  $\tilde{\Psi}(h) : Y_{fp} \rightarrow X_p$ . As a first guess we might try the restriction  $h|_{fp}$  but we see that the codomain of this kernel does not match. Namely we have  $h|_{fp} : Y_{fp} \rightarrow X_{hfp}$  but require a codomain of  $X_p$ . In fact we can verify that  $hfp = p$ : since  $h$  is a Bayesian inverse of  $f$  at  $p$  we have that



and so by the naturality of the delete map:



So it is well defined to set  $\tilde{\Psi}(h) := h|_{fp}$ . We can see that this is an inverse-with-support: to show this we must show that  $\langle h|_{fp} \rangle$  is an ordinary Bayesian inverse. i.e. that



But this follows straightforwardly by two applications of Lemma 25 and the fact that  $h$  is a Bayesian inverse.

We finally have that  $\Psi(-)$  is inverse to  $\tilde{\Psi}$  when viewed as maps to/from equivalence classes:

$$\begin{aligned} \Psi(\tilde{\Psi}(h)) &= i_p \circ r_p \circ h \circ i_{fp} \circ r_{fp} \\ &\simeq_{fp} i_p \circ r_p \circ h \\ &\simeq_{fp} h \end{aligned}$$

where the final equivalence uses the fact that  $h$  is a Bayesian inverse to move it out of the way, similarly to the previous chain of equalities. ◀

Noting that all Bayesian inverses to  $f$  at  $p$  must be  $(f \circ p)$ -almost equal we obtain Theorem 20 as a corollary, that Bayesian inverses with support are unique.

► **Theorem 20 (Unique Bayesian Inversion).** Fix kernels  $f : X \rightarrow Y$  and  $p : I \rightarrow X$ , and support objects  $X_p$  and  $Y_{fp}$ . If  $f$  has an ordinary Bayesian inverse at  $p$ , then there is a unique inverse-with-support to  $f$  at  $p$ .



# Deterministic Constrained Multilinear Detection

Cornelius Brand ✉

Algorithms and Complexity Group, TU Wien, Austria

Viktoriia Korchemna

Algorithms and Complexity Group, TU Wien, Austria

Michael Skotnica

Department of Applied Mathematics, Charles University, Prague, Czech Republic

---

## Abstract

We extend the algebraic techniques of Brand and Pratt (ICALP'21) for deterministic detection of  $k$ -multilinear monomials in a given polynomial with non-negative coefficients to the more general situation of detecting *colored*  $k$ -multilinear monomials that satisfy additional constraints on the multiplicities of the colors appearing in them. Our techniques can be viewed as a characteristic-zero generalization of the algebraic tools developed by Guillemot and Sikora (MFCS'10) and Björklund, Kaski and Kowalik (STACS'13)

As applications, we recover the state-of-the-art deterministic algorithms for the GRAPH MOTIF problem due to Pinter, Schachnai and Zehavi (MFCS'14), and give new deterministic algorithms for generalizations of certain questions on colored directed spanning trees or bipartite planar matchings running in deterministic time  $O^*(4^k)$ , studied originally by Gutin, Reidl, Wahlström and Zehavi (J. Comp. Sys. Sci. 95, '18). Finally, we give improved randomized algorithms for intersecting three and four matroids of rank  $k$  in characteristic zero, improving the record bounds of Brand and Pratt (ICALP'21) from  $O^*(64^k)$  and  $O^*(256^k)$ , respectively, to  $O^*(4^k)$ .

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Fixed-parameter algorithms, Algebraic algorithms, Motif discovery, Matroid intersection

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.25

**Funding** *Cornelius Brand*: Austrian Science Fund (FWF, project Y1329).

*Viktoriia Korchemna*: Austrian Science Fund (FWF, project Y1329).

*Michael Skotnica*: “Grant Schemes at CU” (reg. no. CZ.02.2.69/0.0/0.0/19\_073/0016935), GAČR grant 22-19073S.

## 1 Introduction

The area of fixed-parameter algorithms, sprung from the seminal work of Downey and Fellows (see e.g. their monograph [11]), has produced an enormous amount of tools and techniques to facilitate the design of algorithms that can solve NP-hard problems in running times of the form  $f(k) \cdot \text{poly}(n)$ , where  $n$  is the input size and  $k$  is some parameter that can be interpreted to quantify the difficulty of the instance at hand.

Two prominent and highly successful techniques contributing to this toolbox are, on the one hand, of *algebraic* nature, focusing on formulations of combinatorial problems in the language of polynomials, and then employing mathematical means to solve these reformulations. On the other hand, one of the earliest approaches known to produce fixed-parameter algorithms is the combinatorial method of *representative families*, with their first algorithmic applications dating back at least to work of Monien [25].



© Cornelius Brand, Viktoriia Korchemna, and Michael Skotnica;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics



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

Both the algebraic as well as the representative-family based approach have received a considerable amount of attention in recent years, often focusing on the same application problems. This has resulted in a flurry of competing results on variants of e.g., the notorious longest path problem, subgraph isomorphism, set packing, network design as well as matroid problems, to name a few.

An important variation on the basic combinatorial problems studied in these lines of research concerns *colored* variants of the problems. One of the most well-studied problems of this kind is the *graph motif* problem, which was originally motivated from the analysis of biological networks, and has since been the subject of many parameterized algorithmic studies. As with the uncolored variants of these problems, the colored counterparts attracted attention from researchers from both the perspective of representative families, as well as the algebraic point of view, with both techniques contributing methods that have remained the state-of-the-art within their respective regime. A possibly sweeping generalization of the results that together form this body of work might perhaps conclude that algebraic methods seem more adapted to produce fast randomized algorithms, whereas the representative families tend to yield record bounds for deterministic algorithms.

On the side of algebraic algorithms, in the uncolored regime, a common technique for all of the combinatorial problems above is to formulate them as so-called *multilinear monomial detection* problems. In these problems, one is given an arithmetic circuit computing a polynomial, and the task is to decide whether this polynomial contains a product of  $k$  variables that are all pairwise distinct, and  $k$  is the parameter. Similarly, in the colored variants, these detection problems are generalized to so-called *constrained* multilinear monomial detection, where additional coloring constraints are imposed on the products of variables to be detected. This is also the technique that is considered in the present article.

## 1.1 Related Work

For general background on fixed-parameter algorithms, we refer the reader to the snapshot of the state-of-the-art of the field as captured by e.g. the textbook of Cygan et al. [9], in particular Chapters 10 and 12, where diverse applications of the tools mentioned above are developed.

### Algebraic Algorithms

One of the seminal works for algebraic methods in parameterized and exact algorithms is the work by Björklund et al. [2] on fast subset convolutions and its application for the parameterized Steiner tree problem. More specifically, for the problems considered in this article, the algorithms by Koutis and Williams on multilinear detection were highly influential [20, 22, 29]. These methods were first transported to the setting of constrained multilinear detection by Guillemot and Sikora [17] and subsequently improved by Koutis [21] as well as Björklund, Kowalik and Kaski [4, 5]. It is important to note that all their methods are inherently randomized, because they resort to a use of the DeMillo-Lipton-Schwartz-Zippel Lemma, which makes it seem hard to derandomize them in a black-box manner.

### Graph Problems

These randomized algebraic methods allow to design the state-of-the-art algorithms for e.g. the maximum graph motif problem, running time  $2^k \cdot \text{poly}(n)$  [4, 5]. On the side of deterministic algorithms, which is the focus of this article, the relevant techniques are more combinatorial in nature. Indeed, the method of representative families was first considered

explicitly in the context of fixed-parameter algorithms by Marx [24] and has since been extended into a most intricate machinery, in particular in the works of Fomin et al. [15, 16]. These methods were then refined by Pinter, Shachnai and Zehavi and applied to design the state-of-the-art deterministic algorithms for the graph motif problem [26], running in time  $2^{\omega k} \cdot \text{poly}(n)$ .

## Matroid Problems

Another area in which representative-families based methods have proven fruitful is the realm of *matroid problems*. In this article, we consider the problem of *matroid intersection*: Given  $q$  (representations of) matroids of rank  $k$ , decide whether they share a single common basis. This is a classic problem in combinatorial optimization, and the polynomial-time solvable special case of intersecting  $q = 2$  matroids is famously treated by Edmonds [13]. More generally, intersecting  $q > 2$  matroids becomes *NP*-hard, and fixed-parameter algorithms for this problem were given first (without using this term) by Barvinok [1]. In a later development, Marx [24] revived the interest in this problem by giving the first single-exponential (in  $q$  and  $k$ ) algorithm using representative families, which was superseded by the work of Fomin et al. [15]. The current state-of-the-art is  $4^{qk}$  [8]. A recent manuscript of Eiben, Koana and Wahlström [14] shows that this can be improved to  $4^{(q-2)k}$  using different algebraic methods.

## Our Contribution

The contribution of this paper is three-fold. First, we show how to extend the (deterministic) algebraic machinery of Brand and Pratt to the colored, that is, constrained-multilinear setting. This is noteworthy insofar as until now, the only known algebraic tools for this task were inherently randomized, and the only deterministic algorithms for the respective combinatorial application problems were of decidedly combinatorial nature.

Secondly, we show how to use these adapted methods to, on the one hand, reproduce the deterministic state-of-the-art for the graph motif problem without any additional problem-specific adaptation as a generic application of the algebraic methodology laid out here. In addition, we provide examples of natural colorful extensions of several combinatorial problems, involving e.g. spanning trees and planar perfect matchings, that are not known to admit deterministic algorithms by using only the known combinatorial techniques.

Finally, we improve the state-of-the-art for matroid intersection in the case  $q \leq 4$  by giving specialized polynomial formulations of these cases. It is worth noting that the speedup over the generic state-of-the-art technique is by a factor of 16 (or 4 with respect to [14]) in the exponential base.

## Organization

We continue with a formal introduction of notation and all problems considered in the article. We then prove our main theorem about constrained multilinear detection and give our applications for graph problems. We then conclude with the improved algorithms for matroid problems.

## 2 Preliminaries

Generally, we denote the set of integers  $\{1, \dots, t\}$  by  $[t]$ , and use  $[t]_0$  as a shorthand for  $[t] \cup \{0\}$ . We denote by  $\mathbb{N}$  the set of natural numbers excluding zero, and by  $\mathbb{Q}$  the set of rational numbers.

**Matrices and Matroids**

Let  $A$  be a matrix with row set  $X$  and column set  $Y$ , and let  $X_0$  and  $Y_0$  be subsets of  $X$  and  $Y$  correspondingly. We denote by  $A[X_0, Y_0]$  the submatrix of  $A$  restricted to the rows in  $X_0$  and the columns in  $Y_0$  (while in  $A[Y_0]$  the restriction is applied to columns alone). In particular,  $A[X, Y] = A[Y] = A$ . For two matrices  $A$  and  $B$ , we denote their direct sum by  $A \oplus B$ . For an arbitrary sequence  $r_1, \dots, r_N \in \mathbb{Q}$ , we write  $\text{Vand}_k(r_1, \dots, r_N)$  for the  $k \times N$  *Vandermonde* matrix defined through

$$\text{Vand}_k(r_1, \dots, r_N)_{l,j} = r_j^l, \quad l \in [k - 1]_0, \quad j \in [N].$$

By convention, we let  $0^0 = 1$  in this definition, and we say  $\text{Vand}_k(r_1, \dots, r_N)$  is the Vandermonde matrix of the sequence  $r_1, \dots, r_N$ .

A *finite matroid*  $\mathcal{M}$  is a pair  $(E, \mathcal{I})$ , where  $E$  is a finite set (called the *ground set*) and  $\mathcal{I}$  is a family of subsets of  $E$  (called the *independent sets*) satisfying so-called independence axioms. In this article we only work with finite matroids that can be represented by matrices as follows. Any matrix  $M$  with entries in  $\mathbb{Q}$  gives rise to a matroid  $\mathcal{M}$  with the ground set being its set of columns. The independent sets of the matroid are those subsets of columns that are linearly independent as vectors. In particular, size of any *base* (i.e., maximal with respect to inclusion independent set) of  $\mathcal{M}$  is equal to the rank of  $M$ . We say that such a matroid  $\mathcal{M}$  is *represented* by  $M$ .

**Polynomials**

Let  $R$  be a commutative ring, and let  $x_1, \dots, x_n$  be formal indeterminates. Then  $R[x_1, \dots, x_n]$  is the ring of polynomials in  $x_1, \dots, x_n$  with coefficients in  $R$ , and we call the latter the *coefficient ring* of the polynomial ring.

Every polynomial  $f$  can be represented uniquely as a weighted, finite sum of monomials, that is, products of variables. We may therefore write

$$f = \sum_{a_1, \dots, a_n \in \mathbb{N}} c_{a_1, \dots, a_n} x_1^{a_1} \cdots x_n^{a_n},$$

where only finitely many of the coefficients  $c_{a_1, \dots, a_n} \in R$  are non-zero. When all monomials appearing in  $f_{\mathcal{M}}$  are of degree  $k$ , we call  $f$  itself *homogeneous of degree  $k$* . Furthermore, if for some choice of  $a_1, \dots, a_n \leq 1$  there is a coefficient  $c_{a_1, \dots, a_n} \neq 0$ , we say that  $f$  has a *multilinear* monomial  $x_1^{a_1} \cdots x_n^{a_n}$ .

Our algorithms are based on the algebraic techniques found in [8]. In this approach, as with similar algebraic methods [20], the combinatorial objects (matroid bases, subgraphs, subsets, and so on) over a universe of size  $n$  are modeled using (rational) multivariate polynomials over the indeterminates  $x_1, \dots, x_n$ . The crux of the approach in [8] is then to write down a polynomial whose coefficients encode some information about the combinatorial problem at hand, and evaluate algebraically some linear functional over this polynomial that reveals some sought-after combinatorial answers.

This functional is defined via the following inner product: Let  $f$  and  $g$  be any two such polynomials that are homogeneous of degree  $k$ , and say that  $f$  is as above and

$$g = \sum_{a_1, \dots, a_n \in \mathbb{N}} d_{a_1, \dots, a_n} x_1^{a_1} \cdots x_n^{a_n}.$$

Then we define their *apolar inner product*  $\langle f, g \rangle$  in an almost entirely straightforward way, by

$$\langle f, g \rangle = \sum_{a_1, \dots, a_n \in \mathbb{N}} a_1! \cdot a_2! \cdots a_n! \cdot c_{a_1, \dots, a_n} \cdot d_{a_1, \dots, a_n}. \quad (1)$$

Including the product of factorials serves the purpose of normalization, which allows to connect  $\langle f, g \rangle$  with the partial derivatives of  $f$  and  $g$ .

As an inner product, this mapping can be intuitively interpreted as a measure of similarity between  $f$  and  $g$ . In particular, when fixing  $g$  (or, equivalently,  $f$ ) to, for example, the  $k$ -th elementary symmetric polynomial

$$e_k(x_1, \dots, x_n) = \sum_{\substack{S \subseteq [n] \\ |S|=k}} \prod_{s \in S} x_s,$$

which has all, and only, multilinear monomials with coefficient one, it is easy to see that  $f \mapsto \langle f, e_k \rangle$  is a linear functional that yields the sum of coefficients of multilinear monomials of degree  $k$  of  $f$ . If  $f$  maps to a non-zero value under this functional, we may conclude that  $f$  has a multilinear monomial. As shown in [8], the complexity of evaluating this functional depends on certain algebraic properties of  $f$  and  $e_k$  (or  $g$ , in general).

### Arithmetic Circuits

Even before these algebraic properties, the complexity of this evaluation depends on the *encoding* of  $f$  and  $g$ . Indeed, if  $f$  and  $g$  are given by their list of coefficients, then all questions treated in this article become trivial. However, this would imply inputs that are of exponential size in  $n$ , making this *sparse* encoding a poor choice for polynomials enumerating e.g. combinatorial objects, as an algebraic analog to brute-force search. Instead, polynomials are encoded using *arithmetic circuits*, which are directed acyclic graphs with a single sink, labeled as follows: Every vertex of in-degree zero (*inputs*) is labeled with either an indeterminate or a constant from the coefficient ring. Every vertex with non-zero in-degree is labeled either  $+$  or  $\times$ . The labeled nodes of the arithmetic circuit are referred to as *gates*. An arithmetic circuit *computes* a polynomial in the obvious inductive manner. Finally, we call an arithmetic circuit *skew* if every  $\times$ -gate has at most one edge coming from a non-input gate.

### Constrained Monomials

In the context of this article, multilinear monomials that are not only multilinear, but satisfy additional constraints are relevant. More precisely, suppose  $C = \{1, \dots, q\}$  is a set of  $q$  colors with *multiplicities*  $\mu_1, \dots, \mu_q \in \mathbb{N}$ , and  $\chi : [n] \rightarrow C$  is a *coloring* of  $[n]$ . A multilinear monomial  $x_1^{a_1} \cdots x_n^{a_n}$  is called *well-colored* if  $\sum_{i: \chi(i)=c} a_i \leq \mu_c$  for all colors  $c \in C$ , that is, every color appears at most  $\mu_c$  times in the monomial.

### Problem Statements

Let us now formally introduce the problems studied in this article. The most prominent one, which will be used to reduce the combinatorial application problems to, is the following algebraic problem.

## 25:6 Deterministic Constrained Multilinear Detection

### CONSTRAINED $k$ -MULTILINEAR DETECTION

- Input: A number  $k$ , an arithmetic circuit computing a polynomial  $f$ , a coloring of the variables of  $f$ , together with multiplicity constraints on each color.
- Question: Does  $f$  have a well-colored multilinear monomial of degree  $k$ ?

In our algorithms, we will encounter the restriction that the computed polynomial  $f$  (but not necessarily the circuit itself) have non-negative coefficients in order to make them deterministic. This is a natural restriction when dealing with combinatorial problems in many cases.

### Graph Problems

In analogy to the definition of well-colored monomials, the notion of a vertex or edge coloring includes those mappings that are not necessarily *proper* colorings, that is, two neighboring vertices in a graph, or two edges sharing a vertex, may very well receive the same color. To be precise, given a coloring  $\chi : V(G) \rightarrow \{1, \dots, q\}$ , or analogously  $\chi : E(G) \rightarrow \{1, \dots, q\}$ , and multiplicities  $\mu_1, \dots, \mu_q$ , we call a set  $S$  of vertices (or edges, respectively) *well-colored* if  $\chi^{-1}(i) \cap S \leq \mu_i$  for all  $i = 1, \dots, q$ . With this in mind, the following problems can be defined:

### MAXIMUM GRAPH MOTIF

- Input: A vertex-colored undirected graph  $G$  together with multiplicity constraints on each color.
- Question: Does  $G$  have a well-colored set of  $k$  vertices that induce a connected subgraph of  $G$ ?

In algebraic terms, MAXIMUM GRAPH MOTIF, while amenable to our techniques, doesn't showcase their full strength, for reasons explained in the article. In contrast, the following problems share the important property that they can be expressed succinctly by computations of certain determinants, which allows us to give the first fixed-parameter algorithms for them that are probably hard to come by using other approaches. These are made in analogy to the non-well-colored graph problems studied by Gutin et al. [18].

### WELL-COLORED SPANNING TREE

- Input: A number  $k$  and an edge-colored directed graph  $G$  together with multiplicity constraints on each color.
- Question: Does  $G$  have a well-colored subset of  $k$  edges that can be extended to a directed spanning tree of  $G$ ?

### WELL-COLORED PLANAR PERFECT MATCHING

- Input: A number  $k$  and an edge-colored planar graph  $G$  together with multiplicity constraints on each color.
- Question: Does  $G$  have a well-colored subset of  $k$  edges that can be extended to a perfect matching?

### INTERNALLY WELL-COLORED SPANNING TREE

- Input: A number  $k$  and a vertex-colored planar graph  $G$  together with multiplicity constraints on each color.
- Question: Does  $G$  have a spanning tree such that its internal vertices contain a well-colored subset of at least  $k$  vertices?

### 3 Constrained Multilinear Detection

For any multiplicities  $\mu = (\mu_1, \dots, \mu_q)$  and natural numbers  $n, k$ , consider the polynomial ring  $\mathbb{Q}[y_{1,1}, \dots, y_{1,n}, \dots, y_{q,1}, \dots, y_{q,n}]$  in  $nq$  variables. Following the nomenclature from randomized algebraic methods by Björklund, Kaski and Kowalik [4, 5], we refer to the variables  $y_{i,1}, \dots, y_{i,n}$  as the *shades* of the color  $i$ . We call a multilinear monomial of degree  $k$  in  $y$ -variables *well-colored* if for every color  $i$ , at most  $\mu_i$  shades of the color  $i$  appear in the monomial. Furthermore, we associate with every subset  $M$  of  $[nq]$  of size  $k$  a multilinear monomial  $y_M$  of degree  $k$  in the obvious manner.

► **Lemma 1.** *There is an algorithm that, given  $\mu$ , constructs in time  $\text{poly}(n, k, q)$  a skew arithmetic circuit computing a polynomial  $\chi_\mu = \sum_{M \subseteq [nq], |M|=k} c_M y_M$  such that  $c_M \geq 0$  for all  $M$ , and strict inequality holds if and only if  $y_M$  is well-colored.*

**Proof.** For every  $i \in [q]$ , let

$$\mu_{\leq i} = \sum_{j=1}^i \mu_j$$

be the  $i$ -th partial sum of  $\mu$ . We set  $\mu_{\leq 0} = 0$ . Then, we define a matrix  $S \in \mathbb{Q}^{k \times nq}$  as follows:  $S = (S_1 | S_2 | \dots | S_q)$  is the concatenation of  $q$  blocks  $S_1, \dots, S_q \in \mathbb{Q}^{k \times n}$ , one for each color. Each block  $S_i$  is in turn defined as  $S_i = U_i \cdot V_i$ , where  $U_i \in \mathbb{Q}^{k \times \mu_i}$  is a Vandermonde matrix of dimension  $k \times \mu_i$ , and  $V_i \in \mathbb{Q}^{\mu_i \times n}$  is a Vandermonde matrix of dimension  $\mu_i \times n$ , namely

$$\begin{aligned} U_i &= \text{Vand}_k(\mu_{\leq i-1} + 1, \mu_{\leq i-1} + 2, \dots, \mu_{\leq i}), \\ V_i &= \text{Vand}_{\mu_i}(1, \dots, n). \end{aligned}$$

For a subset  $M \subseteq [nq]$  of size  $k$ , let  $\sigma_M = \det(S[M])$ . We claim that  $\sigma_M \geq 0$  holds, which can be seen as follows. Let  $m = \mu_{\leq q}$ , and consider the auxiliary matrices

$$\begin{aligned} U &= (U_1 | U_2 | \dots | U_q) = \text{Vand}_k(1, \dots, m) \in \mathbb{Q}^{k \times m}, \\ V &= V_1 \oplus \dots \oplus V_q \in \mathbb{Q}^{m \times nq}. \end{aligned}$$

Then per definition, we have  $S = UV$ , and moreover  $S[M] = U \cdot V[M]$ . Therefore, by the Cauchy-Binet formula,

$$\sigma_M = \sum_{\substack{L \subseteq [m] \\ |L|=k}} \det(U[L]) \cdot \det(V[L, M]). \quad (2)$$

Note now that  $U[L]$  is a Vandermonde matrix of an increasing sequence, hence  $\det(U[L])$  is strictly positive, as witnessed by the well-known formula for the Vandermonde determinant:

$$\det(\text{Vand}_k(r_1, \dots, r_N)) = \prod_{1 \leq i < j \leq N} (r_j - r_i).$$

On the other hand, we observe that  $V[L, M]$  is either a direct sum of submatrices of Vandermonde matrices of positive increasing sequences, or has determinant zero. In the former case,  $\det(V[L, M])$  is the product of the determinants of these submatrices. It is also well-known that Vandermonde matrices of positive, increasing sequences are *totally positive*, that is, *all* their minors, not just maximal minors, are positive. Hence, the determinant of

each submatrix is positive, and so is their product, i.e.,  $\det(V[L, M]) > 0$ . Consequently, since we already argued that  $\det(U[L]) > 0$  holds, we have shown that  $\det(V[L, M]) \cdot \det(U[L]) \geq 0$  holds for all  $L$  in (2), and thus in particular  $\sigma_M \geq 0$ .

Furthermore, if  $M$  contains more than  $\mu_i$  indices that belong to the  $i$ -th block of  $S$ , then  $\sigma_M = 0$ : The  $i$ -th block of  $S$  is defined as  $S_i = U_i V_i$  with  $U_i \in \mathbb{Q}^{k \times \mu_i}$  and  $V_i \in \mathbb{Q}^{\mu_i \times n}$ . Hence,  $S_i$  is of rank at most  $\mu_i$ , and any set of more than  $\mu_i$  columns from  $S_i$  will necessarily be linearly dependent. Conversely, if  $M$  contains  $\rho_i \leq \mu_i$  indices belonging to the  $i$ -th block of  $S$  for each  $i$ , pick arbitrary subset  $L$  of rows of  $V$  containing precisely  $\rho_i$  rows from each  $V_i$ . Then  $V[L, M]$  is a direct sum of square submatrices of Vandermonde matrices of increasing sequences, which makes the corresponding term  $\det(U[L]) \det(V[L, M])$  strictly positive. Since we have just argued that all summands in (2) are non-negative, this proves that in this case,  $\sigma_M > 0$ .

Overall, we have shown that  $\sigma_M \neq 0$  if and only if  $M$  contains no more than  $\mu_i$  indices from the same block of  $S$ , which is equivalent to the corresponding monomial  $y_M$  being well-colored. Another application of Cauchy-Binet then provides us with the sought circuit: Letting  $Y$  be the matrix with diagonal entries  $y_{1,1}, y_{1,2}, \dots, y_{n,q}$ , we observe that

$$\det(S \cdot Y \cdot S^T) = \sum_{M \subseteq [nq], |M|=k} \sigma_M^2 \cdot y_M.$$

By the preceding argument, this polynomial has the desired properties demanded in the statement, and the witnessing skew circuit can be written down in polynomial time, using the known constructions for skew determinant circuits [23]. ◀

► **Remark 2.** The proof of the preceding Lemma can also be seen as constructing an explicit representation of the  $k$ -truncation of the partition matroid corresponding to  $\mu$ .

► **Theorem 3.** *There is a deterministic algorithm that, given an  $n$ -variate homogeneous polynomial  $f$  of degree  $k$  with non-negative coefficients, represented as an arithmetic circuit of size  $s$ , as well as multiplicities  $\mu$ , decides in time  $2^{\omega k} \cdot \text{poly}(n, k, s)$  whether or not  $f$  contains a multilinear monomial that is well-colored with respect to  $\mu$ . Here  $\omega < 2.373$  denotes the exponent of matrix multiplication. This running time can be reduced to  $4^k \cdot \text{poly}(n, k, s)$  if the circuit computing  $f$  is skew.*

**Proof.** We invoke [8, Theorem 25] with  $f$  and  $\chi_\mu$  to compute  $\langle f, \chi_\mu \rangle$ . Since both  $f$  (by assumption) and  $\chi_\mu$  (by Lemma 1) have non-negative coefficients, this inner product is zero if and only if no well-colored multilinear monomial exists in  $f$ , and positive otherwise. The claim on the improved running time follows from [8, Theorem 7] and the fact that  $\chi_\mu$  is a determinant polynomial. ◀

## 4 Graph Problems

This theorem allows us to recover the best known bounds for deterministic detection of maximum graph motifs using an entirely different approach.

► **Corollary 4** ([26]). *There is a deterministic algorithm for MAXIMUM GRAPH MOTIF running in time  $2^{\omega k} \text{poly}(n, k)$ .*

**Proof.** All that is needed is a polynomial representation of the set of all  $k$ -vertex connected subgraphs of the input graph. This is possible due to a construction first employed by Guillemot and Sikora [17] on so-called *branching walks*. Consider the following sequence of polynomials:  $P_{i,0} = 1$  for all  $i$ , and

$$P_{i,s} = x_i \sum_{j \in N_G(i)} \sum_{t_1+t_2=s-1} P_{i,t_1} \cdot P_{j,t_2}.$$

The multilinear monomials in  $\beta_k = \sum_{i \in V} P_{i,k}$  can be shown to correspond bijectively to  $k$ -vertex connected subgraphs, and clearly all coefficients are non-negative. Hence, given a coloring  $\chi : [n] \rightarrow C$ , it suffices to evaluate  $\beta_k$  with  $x_j = y_{c,j}$  for all  $j$  such that  $\chi(j) = c$ , for all colors  $c \in C$ . Invoking Theorem 3 then answers whether or not there is a graph motif as sought. ◀

► **Remark 5.** If one were able to design a skew circuit computing  $\beta_k$ , the running time in the preceding theorem would drop immediately to  $4^k$ . From the perspective of algebraic complexity, it is an interesting problem whether such skew circuits for  $\beta_k$  exist, or whether one can rule out their existence under common complexity-theoretic assumptions. The latter could be accomplished e.g. by a completeness proof of  $\beta_k$  for the algebraic complexity class  $VP$ . However, despite heavy research efforts in the past (see e.g. [12] and references therein), very few natural  $VP$ -complete polynomials are known, and the family  $\beta_k$  is not among them.

One conspicuous property of the polynomial  $\beta_k$  is that its computation is *monotone*, that is, no cancellations can arise during its computation. However, the method described here is able to deal also with such cases where cancellations due to negations occur (but the resulting final coefficients are still non-negative). Indeed, this distinction is subtle but crucial: for instance, the determinant can only be computed without using cancellations by circuits of exponential size [19], whereas it is well-known to admit general arithmetic circuits with cancellations of polynomial size. A large number of polynomials that enumerate combinatorial objects can be expressed as determinants, which makes this situation particularly relevant. Moreover, determinants are the prototypical example for polynomials computable by skew circuits, which allows to use the faster running time mentioned in Theorem 3. For instance, this allows us to solve e.g. the following problems in deterministic time  $O^*(4^k)$ :

► **Theorem 6.** *There are deterministic algorithms running in time  $4^k \cdot \text{poly}(n, k)$  for each of the following problems:*

- WELL-COLORED SPANNING TREE,
- WELL-COLORED PLANAR PERFECT MATCHING, and
- INTERNALLY WELL-COLORED SPANNING TREE.

**Proof.** The algorithms for these three problems all follow the same basic principle and build upon the algorithms for the variants where *well-colored* monomials are replaced by monomials with at least  $k$  distinct colors, which is the special case of having multiplicities  $\mu_i = 1$  for all  $i$ . The core idea is to make use of determinantal generating functions for the sought objects in each case. These generating functions are provided by the directed Matrix-Tree theorem and the Pfaffian of planar graphs. Details on these formulations can be found in [6] and [3]. Once these generating functions are available, all that remains to check is that by a standard trick of substituting  $x_i \mapsto (1 + x_i)$  for every variable, these become generating functions for all subsets of solutions (that is, all subsets of edges of spanning trees, all subsets of perfect matchings, etc.), and the claim follows by applying Theorem 3. ◀

## 5 Intersecting Four Matroids

The general method for intersecting  $q$  matroids shown in [8] exploits a well-known connection to matroid parity, and solves the latter problem instead. Indeed, given  $q$  matroids each of rank  $k$  represented by matrices with entries in  $\mathbb{Q}$ , the algorithm in [8] runs in randomized

## 25:10 Deterministic Constrained Multilinear Detection

time  $O^*(4^{kq})$ . In particular, for the cases  $q = 3, 4$ , this specializes to algorithms running in time  $O^*(64^k)$  and  $O^*(256^k)$ , respectively, and [14] improve this to  $O^*(4^k)$  and  $O^*(16^k)$  (and  $O^*(4^{k(q-2)})$  in general). We will now show how to obtain a running time of  $O^*(4^k)$  in both cases, and examine the conditions under which the algorithms can be made deterministic. First, we define an extension of the apolar inner product to tensor products of polynomial rings, that is, polynomial rings that have themselves as a coefficient rings another ring of polynomials. For instance, consider  $f \in \mathbb{Q}[x_1, \dots, x_n] \otimes \mathbb{Q}[y_1, \dots, y_n] \cong \mathbb{Q}[x_1, \dots, x_n][y_1, \dots, y_n]$ . In general,  $f$  has the form

$$f = \sum_{a_1, \dots, a_n \in \mathbb{N}} \sum_{b_1, \dots, b_n \in \mathbb{N}} c_{a_1, \dots, a_n}^{b_1, \dots, b_n} \cdot x_1^{a_1} \cdots x_n^{a_n} \cdot y_1^{b_1} \cdots y_n^{b_n},$$

with only finitely many of the  $c_{a_1, \dots, a_n}^{b_1, \dots, b_n}$  non-zero. Moreover, for fixed  $a_1, \dots, a_n$ , we can collect all the corresponding terms into a single polynomial  $\hat{c}_{a_1, \dots, a_n} \in \mathbb{Q}[y_1, \dots, y_n]$  via

$$\hat{c}_{a_1, \dots, a_n} = \sum_{b_1, \dots, b_n \in \mathbb{N}} c_{a_1, \dots, a_n}^{b_1, \dots, b_n} \cdot y_1^{b_1} \cdots y_n^{b_n},$$

and then recover the familiar

$$f = \sum_{a_1, \dots, a_n \in \mathbb{N}} \hat{c}_{a_1, \dots, a_n} x_1^{a_1} \cdots x_n^{a_n}.$$

While it is true that

$$\mathbb{Q}[x_1, \dots, x_n][y_1, \dots, y_n] \cong \mathbb{Q}[x_1, \dots, x_n, y_1, \dots, y_n],$$

it is important that we distinguish these two ways of looking at  $f$ . In particular, there is nothing new to say about the inner product on the latter polynomial ring, which is well-defined already through Eq. (1). However, for our purposes, we extend the definition Eq. (1) to  $f, g \in \mathbb{Q}[x_1, \dots, x_n][y_1, \dots, y_n]$ , where  $f$  is as above and  $g = \sum_{a_1, \dots, a_n \in \mathbb{N}} \hat{d}_{a_1, \dots, a_n} x_1^{a_1} \cdots x_n^{a_n}$ , with  $\hat{d}_{a_1, \dots, a_n} \in \mathbb{Q}[y_1, \dots, y_n]$  defined analogously to  $\hat{c}_{a_1, \dots, a_n}$ . Then, we set

$$\alpha(f, g) := \sum_{a_1, \dots, a_n \in \mathbb{N}} a_1! \cdots a_n! \cdot \hat{c}_{a_1, \dots, a_n} \cdot \hat{d}_{a_1, \dots, a_n} \in \mathbb{Q}[y_1, \dots, y_n].$$

Note that such a mapping cannot possibly be an inner product anymore (after all, its codomain is not the field  $\mathbb{Q}$ , but  $\mathbb{Q}[y_1, \dots, y_n]$ ), and thus the need for a separate treatment arises. In particular, the algorithms for computing the apolar inner product from [8] do not extend, at least not within the same running time bound, to the case where  $\langle \cdot, \cdot \rangle$  is replaced by  $\alpha$ . However, we can use their results to obtain the following:

► **Lemma 7.** *Let  $f = \det(M_1)$  and  $g = \det(M_2)$ , where  $M_i$  for  $i = 1, 2$  are matrices of dimension  $k \times k$  having bilinear polynomials in  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$  as entries. Then, there is an algorithm that, given  $\bar{y}_1, \dots, \bar{y}_n \in \mathbb{Q}$ , evaluates the polynomial  $\alpha(f, g) \in \mathbb{Q}[y_1, \dots, y_n]$  at  $\bar{y}_1, \dots, \bar{y}_n$  in  $O^*(4^k)$  arithmetic operations over  $\mathbb{Q}$ . Moreover, if the evaluation points  $\bar{y}_1, \dots, \bar{y}_n$  can be encoded using  $O^*(1)$  bits, then  $\alpha(f, g)$  can be evaluated in time  $O^*(4^k)$ .*

**Proof.** By substituting  $\bar{y}_1, \dots, \bar{y}_n$  into  $M_1$  and  $M_2$  and using the fact that evaluation of polynomials at a fixed point is a homomorphism, we find that  $\langle f(\bar{y}_1, \dots, \bar{y}_n), g(\bar{y}_1, \dots, \bar{y}_n) \rangle = \alpha(f, g)(\bar{y}_1, \dots, \bar{y}_n)$  for all  $\bar{y}_1, \dots, \bar{y}_n$ . Since the determinant is well-known to have skew circuits [23], we are in position to apply [8, Theorem 7] to evaluate  $\langle f(\bar{y}_1, \dots, \bar{y}_n), g(\bar{y}_1, \dots, \bar{y}_n) \rangle$  in the required time bound. ◀

► **Theorem 8.** *Let  $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$  be four matroids over a common ground set  $E$  of size  $n$ , each represented by a matrix  $R_i \in \mathbb{Q}^{k \times n}$ . Then, we can decide in randomized time  $O^*(4^k)$  whether these four matroids share a common basis.*

**Proof.** First, let us assume that  $E = [n]$  without loss of generality. We then begin by introducing  $n$  fresh indeterminates  $y_1, \dots, y_n$ . Let then  $Y$  be the diagonal matrix of dimension  $n \times n$  having  $y_i$  in its  $i$ -th diagonal entry. Observe that  $\hat{R}_i := R_i Y$  is the matrix  $R_i$  where the  $i$ -th column was scaled by a factor of  $y_i$ . Moreover, for any set  $S \subset [n]$  of  $k$  column indices, the maximal minor of  $\hat{R}_i$  corresponding to  $S$ , that is, the determinant of the matrix  $\hat{R}_i[S]$  obtained from  $\hat{R}_i$  by restricting to the columns in  $S$ , is a polynomial in  $y_1, \dots, y_n$ . Indeed, since the determinant is a multilinear functional in its columns, we have

$$\det(\hat{R}_i[S]) = \prod_{s \in S} y_s \cdot \det(R_i[S]). \quad (3)$$

In analogy to  $Y$ , let  $X$  be the diagonal matrix having  $x_i$  in its  $i$ -th diagonal entry. Now, the Cauchy-Binet formula gives the following expression for the determinant of the product  $\hat{R}_i \cdot X \cdot R_j^T$ , with a similar reasoning as in Eq. (3):

$$\begin{aligned} \det(\hat{R}_i \cdot X \cdot R_j^T) &= \\ & \sum_{S \subseteq [n], |S|=k} \det(\hat{R}_i[S]) \cdot \det(R_j[S]) \cdot \prod_{s \in S} x_s = \\ & \sum_{S \subseteq [n], |S|=k} \left( \prod_{s \in S} y_s \cdot \det(R_i[S]) \cdot \det(R_j[S]) \right) \cdot \prod_{s \in S} x_s. \end{aligned}$$

In the last line, we grouped the products in order to highlight that we consider this expression foremost as a polynomial in the variables  $x_i$ , that is, an element of  $\mathbb{Q}[y_1, \dots, y_n][x_1, \dots, x_n]$ . Of course, the analogous expression holds for  $\det(R_i \cdot X \cdot R_j^T)$  (note the missing hat on  $R_i$ ), namely

$$\det(R_i \cdot X \cdot R_j^T) = \sum_{S \subseteq [n], |S|=k} \det(R_i[S]) \cdot \det(R_j[S]) \cdot \prod_{s \in S} x_s,$$

which is a polynomial from  $\mathbb{Q}[x_1, \dots, x_n] \subset \mathbb{Q}[y_1, \dots, y_n][x_1, \dots, x_n]$ .

Since by assumption,  $\det(R_i[S]) \neq 0$  if and only if  $S$  is a basis of  $\mathcal{M}_i$ , it follows from the definition of  $\alpha(\cdot, \cdot)$  (and noting that all the  $x_i$  have exponent zero or one) that

$$\alpha(\det(\hat{R}_1 \cdot X \cdot R_2^T), \det(R_3 \cdot X \cdot R_4^T)) = \quad (4)$$

$$\sum_{S \subseteq [n], |S|=k} (\det(R_1[S]) \cdot \det(R_2[S])) \cdot \left( \prod_{s \in S} y_s \cdot \det(R_3[S]) \cdot \det(R_4[S]) \right) = \quad (5)$$

$$\sum_{\substack{S \text{ basis of} \\ \mathcal{M}_1, \dots, \mathcal{M}_4}} \underbrace{\det(R_1[S] R_2[S] R_3[S] R_4[S])}_{\neq 0} \cdot \prod_{s \in S} y_s. \quad (6)$$

As witnessed in the last line of the preceding calculation,  $\alpha(\det(\hat{R}_1 \cdot X \cdot R_2^T), \det(R_3 \cdot X \cdot R_4^T))$  is the zero polynomial if and only if the four input matroids share no common basis. Therefore, all that remains is to test the polynomial  $\alpha(\det(\hat{R}_1 \cdot X \cdot R_2^T), \det(R_3 \cdot X \cdot R_4^T))$  for zero, using the DeMillo–Lipton–Schwartz–Zippel Lemma [10, 30, 28] in combination with Lemma 7, we obtain the desired randomized algorithm: Choosing random evaluation points, we can ensure that  $\alpha(\det(\hat{R}_1 \cdot X \cdot R_2^T), \det(R_3 \cdot X \cdot R_4^T))$  evaluates to a non-zero value, in case

it truly is non-zero, with constant probability. Of course, if the polynomial is identically zero, so is every evaluation, and the algorithm will always correctly recognize this. As usual, repeating this procedure a polynomial number of times allows us to decrease the one-sided error probability exponentially. ◀

► **Corollary 9.** *Given three matroids  $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$  as in Theorem 8, we can decide in randomized time  $O^*(4^k)$  whether they share a common basis.*

**Proof.** This follows from choosing  $\mathcal{M}_1 = \mathcal{M}_2$  in Theorem 8. ◀

► **Remark 1.** During the preparation of the present article, a manuscript by Eiben, Koana and Wahlström [14] appeared, where they give a different algebraic approach to some of the matroid problems considered here. In particular, their Theorem 4.6 coincides with Corollary 9, and they show how to use this as a base case to obtain an algorithm for intersecting  $q$  rank- $k$  matroids, running in time  $4^{k(q-2)} \cdot \text{poly}(n)$ . It would be interesting to see if our Theorem 8 can be expedited in a similar manner to obtain a running time of  $4^{k(q-3)} \cdot \text{poly}(n)$ . Their techniques are based on exterior algebra, which is related to the methods used in this article through a general algebraic connection [7].

## 5.1 Intersecting Positroids

Using the same strategy as for general matroids, we obtain deterministic algorithms for the following important class of matroids:

► **Definition 10.** Let  $k \leq n$ , and let  $\mathcal{M}$  be a matroid over a ground set of size  $n$  of rank  $k$ . Suppose  $\mathcal{M}$  is represented by a matrix  $M$  such that for each submatrix  $M[S]$  with  $S \subset [n]$  and  $|S| = k$  it holds that the corresponding maximal minor satisfies  $\det(M[S]) \geq 0$ . In this case,  $\mathcal{M}$  is called a *positroid*.

It is worth pointing out that while these objects seem not to have made any significant algorithmic appearances so far, they are of great importance in geometry, where they correspond to the so-called *totally non-negative Grassmannian*. They have many desirable properties that general matroids are lacking, most notably a beautiful combinatorial correspondence with certain planar bicolored (or *plabic*) graphs. We refer the reader to Postnikov’s groundbreaking work on the subject [27].<sup>1</sup>

Let us furthermore remark that the following results will only apply to the situation where the input matroids are *promised* to be positroids, since there doesn’t seem to be a way to decide efficiently whether a given matroid representation does indeed only have non-negative minors. While the aforementioned correspondence with plabic graphs does in principle allow for a full-fledged decision problem (by taking as an input not the matrix of the positroid, but its corresponding plabic graph, and then computing a representation from this graph), this is beyond the scope of the present article.

► **Theorem 11.** *Let  $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$  be four positroids over a common ground set  $E$  of size  $n$ , each represented by a matrix  $R_i \in \mathbb{Q}^{k \times n}$  with non-negative minors. Then, we can decide in deterministic time  $O^*(4^k)$  whether these four matroids share a common basis.*

**Proof.** The proof relies on the fact that all determinants in Eq. (6) are not only non-zero, but in fact positive. Therefore, it is not necessary to include the variables  $y_i$  in the calculation, and a direct application of [8, Theorem 7] to  $(\det(R_1 \cdot X \cdot R_2^T), \det(R_3 \cdot X \cdot R_4^T))$  is already enough. The resulting value is non-zero if and only if the four positroids share a common basis, and the running time bound follows directly from [8]. ◀

<sup>1</sup> This particular manuscript, despite being cited hundreds of times, didn’t appear in a journal.

---

**References**

---

- 1 Alexander I Barvinok. New algorithms for linear k-matroid intersection and matroid k-parity problems. *Mathematical Programming*, 69:449–470, 1995.
- 2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.
- 3 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed hamiltonicity and out-branchings via generalized laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 91:1–91:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.91.
- 4 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Probably optimal graph motifs. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPICs*, pages 20–31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.STACS.2013.20.
- 5 Andreas Björklund, Petteri Kaski, and Lukasz Kowalik. Constrained multilinear detection and generalized graph motifs. *Algorithmica*, 74(2):947–967, 2016. doi:10.1007/s00453-015-9981-1.
- 6 Cornelius Brand. Patching colors with tensors. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.25.
- 7 Cornelius Brand. A note on algebraic techniques for subgraph detection. *Inf. Process. Lett.*, 176:106242, 2022. doi:10.1016/j.ipl.2021.106242.
- 8 Cornelius Brand and Kevin Pratt. Parameterized applications of symbolic differentiation of (totally) multilinear polynomials. 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 38:1–38:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.38.
- 9 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.
- 10 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.
- 11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 12 Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for VP. *Chic. J. Theor. Comput. Sci.*, 2016, 2016. URL: <http://cjtcs.cs.uchicago.edu/articles/2016/3/contents.html>.
- 13 Jack Edmonds. Matroid intersection. In *Annals of discrete Mathematics*, volume 4, pages 39–49. Elsevier, 1979.
- 14 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. arXiv:2304.02091.
- 15 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

- 16 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative families of product families. *ACM Trans. Algorithms*, 13(3):36:1–36:29, 2017. doi:10.1145/3039243.
- 17 Sylvain Guillemot and Florian Sikora. Finding and counting vertex-colored subtrees. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 405–416. Springer, 2010. doi:10.1007/978-3-642-15155-2\_36.
- 18 Gregory Z. Gutin, Felix Reidl, Magnus Wahlström, and Meirav Zehavi. Designing deterministic polynomial-space algorithms by color-coding multivariate polynomials. *J. Comput. Syst. Sci.*, 95:69–85, 2018. doi:10.1016/j.jcss.2018.01.004.
- 19 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, 1982. doi:10.1145/322326.322341.
- 20 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Track A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008. doi:10.1007/978-3-540-70575-8\_47.
- 21 Ioannis Koutis. Constrained multilinear detection for faster functional motif discovery. *Inf. Process. Lett.*, 112(22):889–892, 2012. doi:10.1016/j.ipl.2012.08.008.
- 22 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009. doi:10.1007/978-3-642-02927-1\_54.
- 23 Meena Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In Michael E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA*, pages 730–738. ACM/SIAM, 1997. URL: <http://dl.acm.org/citation.cfm?id=314161.314429>.
- 24 Dániel Marx. A parameterized view on matroid optimization problems. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 655–666. Springer, 2006. doi:10.1007/11786986\_57.
- 25 Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.
- 26 Ron Y. Pinter, Hadas Shachnai, and Meirav Zehavi. Deterministic parameterized algorithms for the graph motif problem. 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 II*, volume 8635 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2014. doi:10.1007/978-3-662-44465-8\_50.
- 27 Alexander Postnikov. Total positivity, grassmannians, and networks, 2006. arXiv:math/0609764.
- 28 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.
- 29 Ryan Williams. Finding paths of length  $k$  in  $o^*(2^k)$  time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.
- 30 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposium on Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5\_73.

# Rational Verification for Nash and Subgame-Perfect Equilibria in Graph Games

Léonard Brice ✉

Université Libre de Bruxelles, Belgium

Jean-François Raskin ✉

Université Libre de Bruxelles, Belgium

Marie van den Bogaard ✉

Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

---

## Abstract

We study a natural problem about rational behaviors in multiplayer non-zero-sum sequential infinite duration games played on graphs: rational verification, that consists in deciding whether all the rational answers to a given strategy satisfy some specification. We give the complexities of that problem for two major concepts of rationality: Nash equilibria and subgame-perfect equilibria, and for three major classes of payoff functions: energy, discounted-sum, and mean-payoff.

**2012 ACM Subject Classification** Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

**Keywords and phrases** Games on graphs, Nash equilibria, subgame-perfect equilibria

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.26

**Related Version** *Full Version*: <https://arxiv.org/abs/2301.12913>

## 1 Introduction

Formal methods are essential to guarantee the correctness of safety critical computer systems. Techniques like model-checking [13] or automated theorem proving [15] are now routinely used to develop systematically hardware pieces as well as embedded control systems. Nevertheless, there are contexts in which formal methods have not yet been applied successfully large-scale: that is the case of multi-agent systems, which still represent a challenge for formal verification techniques, because they are usually composed of *heterogeneous* components, ranging from traditional pieces of reactive code to wholly autonomous robots or human users. Producing operational model abstractions for this diversity of sub-systems is often challenging.

While it may be inconvenient, to say the least, to produce an operational model of the behavior of a human or a complex autonomous robot, identifying the high level objectives of those components may be easier. Taking into account those objectives is often key for reasoning about the correctness of a system that interacts with those components. Indeed, a system is usually not supposed to be correct in all circumstances, but only when agents in its environment behave in a way that concurs with their own objectives. In *rational verification* (RV), a system needs to enforce some property, not in all possible executions, but only in those in which the environment agents behave rationally with regards to their own objectives.

Rationality is the focus point of game theory, and can be formalized in several ways: for instance, with the notion of *Nash equilibrium* (NE) [24]. NEs have been used in a few promising contributions, like in verification of non-repudiation and fair exchange protocols [12, 20, 21], or planning of self-driving cars interacting with human drivers [26], etc. Nevertheless, those works do not propose a general framework for RV and their contributions are rather specific to their application domains. There is thus a need for more systematic study of formal frameworks for RV. Such a study has been started recently: for instance, the authors of [18]



© Léonard Brice, Jean-François Raskin, and Marie van den Bogaard;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 26; pp. 26:1–26:15

Leibniz International Proceedings in Informatics



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

study the automatic verification of an LTL specification in multi-agent systems that behave according to an NE, and in [11], the authors study a setting in which the environment has multiple objectives and only produces behaviors that are Pareto-optimal with regards to them. This work contributes to that line of research by considering a notion of rationality formalized by *subgame-perfect equilibria* (SPEs), a refinement of NEs that is better suited to formalize rationality in sequential games, in which NEs suffer from non-credible threats [25].

More precisely, we consider here the *rational verification problem*, which takes as inputs: (i) a multiplayer game graph with a designated player called *Leader*, (ii) a finite state description of a (potentially infinite) set of strategies for Leader, (iii) a description of the objective for Leader, and (iv) a description of the objectives of all the other players. It asks whether for all possible fixed strategies  $\sigma_L$  of Leader (defined by the finite state description), for all possible rational responses of the other agents, the generated outcome satisfies Leader's objective. That problem is well-suited to formalize the verification of correctness of a controller interacting with an environment composed of rational agents.

**Contributions.** To solve the RV problems, we provide a general construction, called the *product game* (Definition 5): we show that, given a game and a finite-state description of a set of Leader's strategies, one can incorporate the memory states of that finite-state description in the arena of the game in a way that Leader is implicitly forced to follow some strategy in the set. Thus, we show that the RV problem reduces in polynomial time to the *universal threshold problem*, a problem that is easier to study algorithmically: given a game, does every equilibrium satisfy a given specification? Also, some game classes we analyze have been addressed with slightly different definitions in previous literature. Interestingly, we provide a reduction in the opposite direction as well (Cor. 6).

We use that tool to prove the undecidability of RV in energy games (Th. 9 and 10); in the case of subgame-perfect RV, we show that undecidability holds even when Leader plays against only two players. We show that Nash RV is co-recursively enumerable in those games, and leave that question open for subgame-perfect RV – but contrary to the Nash setting, SPEs may require infinite memory to reach some payoffs (Prop. 11). In discounted-sum (DS) games, we show that the RV problems are at least as hard as the *target discounted-sum problem* (Th. 13), whose decidability is an open question. However, we prove that those problems are recursively enumerable (Th. 14). In the case of mean-payoff (MP) games, Cor. 8, combined with older results, entails that the RV problems are coNP-complete. But that case highlights a subtlety in the definition of RV: if one wants to check that a strategy is such that *every* rational response satisfies the specification, then when no such response exists, the strategy will be accepted. In the case of MP games, that leads to results that can be considered as counter-intuitive. We thus propose a stronger definition of the RV problem, called *chaotic RV*, to avoid that weakness: it consists in deciding whether a strategy satisfies the specification against every response that is *as rational as it can be*, using the notions of  $\varepsilon$ -NE and  $\varepsilon$ -SPE, that are quantitative relaxations of NE and SPE. We show that such a problem is  $\text{P}^{\text{NP}}$ -complete in MP games (Th. 19), and that in every other setting (Nash or subgame-perfect RV in the two other game classes), it coincides with RV, since rational responses always exist (Prop. 17). A synopsis of those results can be found in Table 1.

**Related works.** During the last decade, multiplayer games and their applications to reactive synthesis have raised a growing attention: the reader may refer to [3, 9, 10, 16, 22], and their references. The concept of *rational verification* appears in [19], where Gutierrez, Najib, Perelli, and Wooldridge give the complexity of several related problems. They use a definition

■ **Table 1** Synopsis of our results.

	Nash RV	Ach. Nash RV	SP RV	Ach. SP RV
Energy	undecidable, co-RE		undecidable	
DS	TDS-hard, RE		TDS-hard, RE	
MP	coNP-complete		coNP-complete	P <sup>NP</sup> -complete

that is slightly different from ours: their problem consists in deciding, given a game and a specification, whether all NEs (or one of them) in that game satisfy the specification, without any player representing the system (Leader in our setting). Still, as we show with Cor. 8, that problem is strongly related to ours. In [27], they also study if  $\omega$ -regular properties are enforced by NEs induced by mean-payoff objectives. The objectives considered in those papers are only  $\omega$ -regular objectives. Moreover, both in [19] and in [27] only NEs are considered, while our main contributions are about SPEs, that are arguably better suited for reasoning about sequential games [25], but also require substantially more complex techniques. In [14], Filiot, Gentilini, and Raskin study *Stackelberg values* of mean-payoff and discounted-sum two-player non-zero sum games, i.e. the payoff that Leader gets when the other player, *Follower*, plays the *best response* that is available with regards to his own objective. This is a synthesis problem while we consider a verification problem. They consider only one player in the environment while we consider the more general case of  $n$  players.

In [28], and later in [29], Ummels studies SPEs in parity games. He proves that they always exist, and that deciding whether there exists an SPE in a given parity game that generates a payoff vector between two given thresholds (the *constrained existence problem*, very close to the *universal threshold problem* studied in this paper) is EXPTIME-easy and NP-hard. In [8], Brihaye, Bruyère, Goeminne, Raskin, and van den Bogaard, study the same problem in quantitative reachability games, and prove that it is PSPACE-complete.

In [17], Flesch and Predtetchinski give a general procedure to characterize SPEs. In [4], Brice, Raskin, and van den Bogaard introduce the *negotiation function*, a tool that turns Flesch and Predtetchinski's procedure into effective algorithms for a large class of games. In [6], they use it to close the gap left by Ummels, proving that the constrained existence problem is NP-complete in parity games, with methods that they use later in [5] to prove that the same problem is also NP-complete in mean-payoff games. An alternative procedure to solve such SPE problems is proposed in [23], where Meunier constructs a two-player zero-sum game in which one player has a winning strategy if and only if there exists an SPE satisfying the desired constraint in the input game. That technique is nevertheless often costly, because the size of the constructed game is proportional to the number of possible payoff vectors; and for the same reason, it cannot be applied to games with infinite payoff spaces.

Energy objectives have also been widely studied, in connection with the study of vector additions systems with states and Petri nets, but almost always in a two-player zero-sum setting: see for instance [2, 22, 30]. As for discounted-sum objectives, they are defined for instance by Zwick and Paterson in [31], again in a two-player zero-sum setting. They are strongly related to the target discounted-sum problem, which is a long-standing open problem, as shown in [1] by Boker, Henzinger, and Otop. To the best of our knowledge, no algorithmic results are known for those classes of objectives in a multiplayer non-zero sum setting.

**Structure of the paper.** In Sec. 2, we introduce the necessary background. In Sec. 3, we present the product game. In Sec. 4, we exploit it to study energy games; in Sec. 5, DS games; and in Sec. 6, MP games. The complete proofs of our results, and additional results, are given in the complete version of this paper [7].

## 2 Background

**Graphs, games and strategies.** We call *graph* a finite directed graph, i.e. a pair  $(V, E)$  where  $V$  is a finite set of *vertices* and  $E \subseteq V \times V$  is a set of *edges*. The edge  $(u, v)$ , written  $uv$ , is an *outgoing edge* of  $u$ . A *path* in  $(V, E)$  is a finite or infinite sequence  $\alpha = \alpha_0\alpha_1 \cdots \in V^* \cup V^\omega$  such that for every index  $k$ , we have  $\alpha_k\alpha_{k+1} \in E$ . We write  $\text{Occ}(\alpha)$  (resp.  $\text{Inf}(\alpha)$ ) for the set of vertices that occur (resp. that occur infinitely often) in  $\alpha$ . For a given index  $k$ , we write  $\alpha_{\leq k} = \alpha_{<k+1} = \alpha_0 \dots \alpha_k$ , and  $\alpha_{\geq k} = \alpha_{>k-1} = \alpha_k\alpha_{k+1} \dots$ . A *cycle* is a finite path  $c = c_0 \dots c_n$  with  $c_n c_0 \in E$ . A finite path  $\alpha$  is *simple* if for every two indices  $k \neq \ell$ , we have  $\alpha_k \neq \alpha_\ell$ .

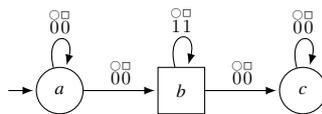
We call *non-initialized game* a tuple  $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, E, \mu)$ , where:

- $\Pi$  is a finite set of *players*;
- $(V, E)$  is a graph, in which every vertex has at least one outgoing edge;
- $(V_i)_{i \in \Pi}$  is a partition of  $V$ , in which  $V_i$  is the set of vertices *controlled* by player  $i$ ;
- a *play* (resp. *history*) in the game  $\mathcal{G}$  is an infinite (resp. finite) path in the graph  $(V, E)$ , and the set of plays (resp. histories) in  $\mathcal{G}$  is denoted by  $\text{Plays}\mathcal{G}$  (resp.  $\text{Hist}\mathcal{G}$ );
- the *payoff function*  $\mu : \text{Plays}\mathcal{G} \rightarrow \mathbb{R}^\Pi$  maps each play  $\pi$  to the tuple  $\mu(\pi) = (\mu_i(\pi))_{i \in \Pi}$ .

Given a set of players  $P \subseteq \Pi$ , we often write  $V_P = \bigcup_{i \in P} V_i$ . When  $i$  is a player and when the context is clear, we write  $-i$  for the set  $\Pi \setminus \{i\}$ . We often assume that a special player, called *Leader* and denoted by the symbol  $\mathbb{L}$ , belongs to the set  $\Pi$ . An *initialized game* is a pair  $(\mathcal{G}, v_0)$ , often written  $\mathcal{G}_{\uparrow v_0}$ , where  $\mathcal{G}$  is a non-initialized game and  $v_0 \in V$  is a vertex called *initial vertex*. When the context is clear, we use the word *game* for both initialized and non-initialized games. A play (resp. history) in the initialized game  $\mathcal{G}_{\uparrow v_0}$  is a play (resp. history) that has  $v_0$  as first vertex. The set of plays (resp. histories) in  $\mathcal{G}_{\uparrow v_0}$  is denoted by  $\text{Plays}\mathcal{G}_{\uparrow v_0}$  (resp.  $\text{Hist}\mathcal{G}_{\uparrow v_0}$ ). We also write  $\text{Hist}_i\mathcal{G}$  (resp.  $\text{Hist}_i\mathcal{G}_{\uparrow v_0}$ ) for the set of histories in  $\mathcal{G}$  (resp.  $\mathcal{G}_{\uparrow v_0}$ ) whose last vertex is controlled by player  $i$ .

A *strategy* for player  $i$  in the initialized game  $\mathcal{G}_{\uparrow v_0}$  is a mapping  $\sigma_i : \text{Hist}_i\mathcal{G}_{\uparrow v_0} \rightarrow V$ , such that  $v\sigma_i(hv)$  is an edge of  $(V, E)$  for every  $hv$ . A history  $h$  is *compatible* with a strategy  $\sigma_i$  if and only if  $h_{k+1} = \sigma_i(h_0 \dots h_k)$  for all  $k$  such that  $h_k \in V_i$ . This definition naturally extends to plays. A *strategy profile* for  $P \subseteq \Pi$  is a tuple  $\bar{\sigma}_P = (\sigma_i)_{i \in P}$ , where each  $\sigma_i$  is a strategy for player  $i$  in  $\mathcal{G}_{\uparrow v_0}$ . A play, or a history, is *compatible* with  $\bar{\sigma}_P$  if it is compatible with every  $\sigma_i$  for  $i \in P$ . Since the  $\sigma_i$ 's domains are pairwise disjoint, we sometimes consider  $\bar{\sigma}_P$  as one function: for  $hv \in \text{Hist}\mathcal{G}_{\uparrow v_0}$  such that  $v \in \bigcup_{i \in P} V_i$ , we liberally write  $\bar{\sigma}_P(hv)$  for  $\sigma_i(hv)$  with  $i$  such that  $v \in V_i$ . A *complete* strategy profile, usually written  $\bar{\sigma}$ , is a strategy profile for  $\Pi$ . Exactly one play is compatible with the strategy profile  $\bar{\sigma}$ : we call it its *outcome* and write  $\langle \bar{\sigma} \rangle$  for it. When  $\bar{\tau}_P$  and  $\bar{\tau}'_Q$  are two strategy profiles with  $P \cap Q = \emptyset$ , we write  $(\bar{\tau}_P, \bar{\tau}'_Q)$  for the strategy profile  $\bar{\sigma}_{P \cup Q}$  such that  $\sigma_i = \tau_i$  for  $i \in P$ , and  $\sigma_i = \tau'_i$  for  $i \in Q$ .

**Notable classes of games.** Here, we will focus on three game classes. In those classes, each player  $i$ 's payoff is based on a *reward mapping*  $r_i : E \rightarrow \mathbb{Q}$ . Intuitively, the reward mapping gives the (positive or negative) reward that player  $i$  gets for each action. The first class, energy games, is a class of *Boolean games*, i.e. games in which all payoffs are equal either to 0 or to 1. For such games, we say that player  $i$  *loses* the play  $\pi$  when  $\mu_i(\pi) = 0$ , and *wins* it when  $\mu_i(\pi) = 1$ . The other games are called *quantitative*. In energy games, the players seek to keep the aggregated sum of those rewards, their *energy level*, always nonnegative. That quantity symbolizes any resource that an agent could have to store: fuel, money, ...



■ **Figure 1** An example of mean-payoff game.

► **Definition 1** (Energy). *In a graph  $(V, E)$ , we associate to each reward mapping  $r$  the energy level function  $\text{EL}_r : \text{Hist}\mathcal{G} \rightarrow \mathbb{N} \cup \{\perp\}$  defined by:*

- $\text{EL}_r(h_0) = 0$ ;
- $\text{EL}_r(h_{\leq n+1}) = \text{EL}_r(h_{\leq n}) + r(h_n h_{n+1})$  if  $\text{EL}_r(h_{\leq n}) \neq \perp$ , and  $\text{EL}_r(h_{\leq n}) + r(h_n h_{n+1}) \geq 0$ ;
- $\text{EL}_r(h_{\leq n+1}) = \perp$  otherwise.

*The game  $\mathcal{G}$  is an energy game if there exists a tuple  $(r_i)_{i \in \Pi}$  of reward mappings such that for each  $i$  and every  $\pi$ , we have  $\mu_i(\pi) = 0$  if  $\text{EL}_{r_i}(\pi_{\leq n}) = \perp$  for some  $n$ , and  $\mu_i(\pi) = 1$  otherwise. When the context is clear, we write  $\text{EL}_i$  for  $\text{EL}_{r_i}$ .*

In discounted-sum games, each player's payoff is obtained by summing the rewards that the player obtains with some discount factor applied as the play goes along.

► **Definition 2** (Discounted-sum). *In a graph  $(V, E)$ , we define for each reward mapping  $r$  and each discount factor  $\lambda \in (0, 1)$  the discounted sum function  $\text{DS}_r^\lambda : h \mapsto \sum_k \lambda^k r(h_k h_{k+1})$ . Then, we write  $\text{DS}_r^\lambda(\pi) = \lim_n \text{DS}_r^\lambda(\pi_{\leq n})$ . The game  $\mathcal{G}$  is a discounted-sum game (or DS game for short) if there exists a discount factor  $\lambda \in (0, 1) \cap \mathbb{Q}$  and a tuple  $(r_i)_{i \in \Pi}$  of reward mappings such that for each  $i$  and every  $\pi$ , we have  $\mu_i(\pi) = \text{DS}_{r_i}^\lambda(\pi)$ . When the context is clear, we write  $\text{DS}_i$  for  $\text{DS}_{r_i}^\lambda$ .*

In mean-payoff games, a players' payoff is equal to their asymptotic average reward.

► **Definition 3** (Mean-payoff). *In a graph  $(V, E)$ , we define for each reward mapping  $r$  the mean-payoff function  $\text{MP}_r : h_0 \dots h_n \mapsto \frac{1}{n} \sum_k r(h_k h_{k+1})$ . Then, we write  $\underline{\text{MP}}_r(\pi) = \liminf_n \text{MP}_r(\pi_{\leq n})$ . The game  $\mathcal{G}$  is a mean-payoff game (or MP game for short) if there exists a tuple  $(r_i)_{i \in \Pi}$  of reward mappings, such that for each player  $i$ , we have  $\mu_i = \underline{\text{MP}}_{r_i}$ . When the context is clear, we write  $\text{MP}_i$  for  $\text{MP}_{r_i}$ , and  $\underline{\text{MP}}_i$  for  $\underline{\text{MP}}_{r_i}$ .*

Every game  $\mathcal{G}$  from one of those three classes can be encoded with a finite number of bits. We write  $\|\mathcal{G}\|$  for that number.

An example of MP game is given in Figure 1, with two players: player  $\circ$ , who controls the vertices  $a$  and  $c$ , and player  $\square$ , who controls the vertex  $b$ . The initial vertex is  $v_0 = a$ . We wrote above each edge the rewards that both players get when that edge is taken. Three types of plays are possible in that game: the one that loops on the vertex  $a$  gives both players the payoff 0; the ones that loop on the vertex  $b$  give both players the payoff 1; and the ones that loop on the vertex  $c$  give both players the payoff 0.

**Equilibria and rational responses.** In this paper, we study rational behaviors of players: we have, therefore, to define our rationality concepts. Let us start with the most classical one: *Nash equilibrium*. The strategy profile  $\bar{\sigma}$  is a *Nash equilibrium* (resp.  $\mathbb{L}$ -fixed Nash equilibrium) – or  $\mathbb{L}$ -fixed NE for short – in  $\mathcal{G}_{\uparrow v_0}$  if for each player  $i$  (resp. each player  $i \neq \mathbb{L}$ ) and every strategy  $\sigma'_i$ , called *deviation of  $\sigma_i$* , we have  $\mu_i(\langle \sigma'_i, \bar{\sigma}_{-i} \rangle) \leq \mu_i(\langle \bar{\sigma} \rangle)$ . When it is not the case, we call *profitable deviations* the deviations that do not satisfy that inequality.

As an example, in the game given in Figure 1, two types of NEs can be found: those that eventually loop on the vertex  $b$ , and give both players the payoff 1; and those that loop on  $a$ , but in which player  $\circ$  has no profitable deviation, because if she goes to the vertex  $b$ , player

□ threatens to go to the vertex  $c$  (and player □ has no profitable deviation, because he does never make any choice). However, player □'s threat is not *credible*, since going to the vertex  $c$  would give him the payoff 0, while he could stay on the vertex  $b$  and get the payoff 1. A stronger rationality concept, that avoids that phenomenon, is the one of *subgame-perfection*.

Let  $hv$  be a history in the game  $\mathcal{G}$ . The *subgame* of  $\mathcal{G}$  after  $hv$  is the game  $\mathcal{G}_{\uparrow hv} = (\Pi, V, (V_i)_i, E, \mu_{\uparrow hv})_{\uparrow v}$ , where  $\mu_{\uparrow hv}$  maps each play  $\pi$  to its payoff in  $\mathcal{G}$ , assuming that the history  $hv$  has already been played, i.e. to the payoff  $\mu_{\uparrow hv}(\pi) = \mu(h\pi)$ . If  $\sigma_i$  is a strategy in  $\mathcal{G}_{\uparrow v_0}$ , its *substrategy* after  $hv$  is the strategy  $\sigma_{i\uparrow hv} : h' \mapsto \sigma_i(hh')$  in the game  $\mathcal{G}_{\uparrow hv}$ .

The strategy profile  $\bar{\sigma}$  is a ( $\mathbb{L}$ -fixed) *subgame-perfect equilibrium* – or ( $\mathbb{L}$ -fixed) *SPE* for short – in  $\mathcal{G}_{\uparrow v_0}$  if and only if for every history  $h$  in  $\mathcal{G}_{\uparrow v_0}$  (resp. every history  $h$  compatible with  $\sigma_{\mathbb{L}}$ ), the strategy profile  $\bar{\sigma}_{\uparrow h}$  is a ( $\mathbb{L}$ -fixed) Nash equilibrium in the subgame  $\mathcal{G}_{\uparrow h}$ .

NEs and SPEs entail two notions of rationality for the environment's responses to a strategy  $\sigma_{\mathbb{L}}$  of Leader. A strategy profile  $\bar{\sigma}_{-\mathbb{L}}$  is a *Nash response* to  $\sigma_{\mathbb{L}}$  if the strategy profile  $\bar{\sigma} = (\sigma_{\mathbb{L}}, \bar{\sigma}_{-\mathbb{L}})$  is an  $\mathbb{L}$ -fixed NE, and a *subgame-perfect response* if it is an  $\mathbb{L}$ -fixed SPE. The set of Nash (resp. subgame-perfect) responses to  $\sigma_{\mathbb{L}}$  is written  $\text{NR}(\sigma_{\mathbb{L}})$  (resp.  $\text{SPR}(\sigma_{\mathbb{L}})$ ).

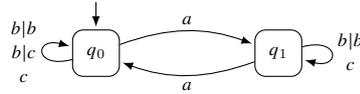
Finally, let  $\rho \in \{\text{Nash}, \text{subgame-perfect}\}$ . We call  $\rho$ -*equilibria* the NEs if  $\rho = \text{Nash}$ , and the SPEs if  $\rho = \text{subgame-perfect}$ . We will similarly talk about  $\mathbb{L}$ -fixed  $\rho$ -*equilibria*, and  $\rho$ -*responses*. We write  $\rho\text{R}(\sigma_{\mathbb{L}})$  for the set of  $\rho$ -responses to a strategy  $\sigma_{\mathbb{L}}$ .

**Mealy machines.** A *Mealy machine* for player  $i$  on a game  $\mathcal{G}$  is a tuple  $\mathcal{M} = (Q, q_0, \Delta)$ , where  $Q$  is a finite set of *states*, where  $q_0 \in Q$  is the *initial state*, and where  $\Delta \subseteq (Q \times V_{-i} \times Q) \cup (Q \times V_i \times Q \times V)$  is a finite set of *transitions*, such that for every  $(p, u, q, v) \in \Delta$ , we have  $uv \in E$ , and such that for every  $p \in Q$  and  $u \in V$ , there exists a transition  $(p, u, q)$  or  $(p, u, q, v) \in \Delta$ . Specialist readers will have noted that this definition is more general than the classical one, in which it is often assumed that for each  $p$  and  $u$ , there exists exactly one such transition: hereafter, such a machine will be called *deterministic*. Results about deterministic Mealy machines can be applied to *programs*, which are supposed to run deterministically; we chose to take a more general definition to capture also *protocols*, which may be given to an agent who would still have some room for manoeuvre in how they apply it.

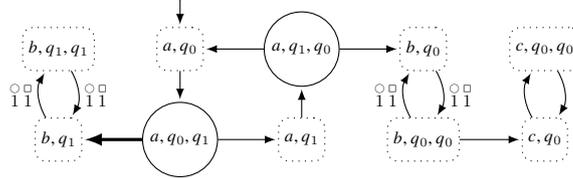
A strategy  $\sigma_i$  in  $\mathcal{G}_{\uparrow v_0}$  is *compatible* with  $\mathcal{M}$  if there exists a mapping  $h \mapsto q_h$  that maps every history  $h$  in  $\mathcal{G}_{\uparrow v_0}$  to a state  $q_h \in Q$ , such that for every  $hv \in \text{Hist}_{-i}\mathcal{G}_{\uparrow v_0}$ , we have  $(q_h, v, q_{hv}) \in \Delta$ , and for every  $hv \in \text{Hist}_i\mathcal{G}_{\uparrow v_0}$ , we have  $(q_h, v, q_{hv}, \sigma_i(hv)) \in \Delta$ . The set of strategies in  $\mathcal{G}_{\uparrow v_0}$  compatible with  $\mathcal{M}$  is written  $\text{Comp}_{\uparrow v_0}(\mathcal{M})$ . If  $\mathcal{M}$  is deterministic, then there is exactly one strategy compatible with  $\mathcal{M}$ ; we call it a *finite-memory* strategy.

Note that one can define analogously Mealy machines that capture a set of strategy profiles for several players, and even for the whole set  $\Pi$ . Note also that every Mealy machine  $\mathcal{M}$  can be encoded with a finite number of bits: we write  $\|\mathcal{M}\|$  for that number.

Figure 2 depicts a Mealy machine on the game of Figure 1. Each arrow from a state  $p$  to a state  $q$  labeled  $u|v$  denotes the existence of a transition  $(p, u, q, v)$  (from the state  $p$ , the machine reads the vertex  $u$ , switches to the state  $q$  and outputs the vertex  $v$ ). Each arrow from a state  $p$  to a state  $q$  labeled  $u$  denotes the existence of a transition  $(p, u, q)$  (from  $p$ , the machine reads  $u$ , switches to  $q$  and outputs nothing). It is a machine for player □, that is not deterministic: from the state  $q_0$ , reading the vertex  $b$ , the machine stays in  $q_0$  but it can output either  $b$  or  $c$ . The strategies that are compatible with it can be described as follows: when player □ has to play, if the vertex  $a$  was seen an odd number of times, then he stays in  $b$ ; in the opposite case, he can either stay in  $b$  or eventually go to  $c$ .



■ **Figure 2** A non-deterministic one-player Mealy machine.



■ **Figure 3** A product game.

**Decision problem.** Let us now define rational verification (RV). We define it for each game class  $\mathcal{C}$ , for each  $\rho \in \{\text{Nash, subgame-perfect}\}$ , and in both the deterministic and the non-deterministic setting.

► **Problem 4** ((Deterministic)  $\rho$ -rational verification problem in the class  $\mathcal{C}$ ). *Given a game  $\mathcal{G}_{\uparrow v_0} \in \mathcal{C}$ , a threshold  $t \in \mathbb{Q}$  and a (deterministic) Mealy machine  $\mathcal{M}$  on  $\mathcal{G}$ , is every  $\mathbb{L}$ -fixed  $\rho$ -equilibrium  $\bar{\sigma}$  with  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$  such that  $\mu_{\mathbb{L}}(\langle \bar{\sigma} \rangle) > t$ ?*

### 3 The product game

Although very intuitive, the RV problems are quite hard to study as they are. Indeed, their instances include two graph structures: a game and a Mealy machine. However, responding rationally to Leader's strategies that are compatible with  $\mathcal{M}$  amounts to play rationally in a larger game, in which the machine  $\mathcal{M}$  has been incorporated.

► **Definition 5** (Product game). *Let  $\mathcal{G}_{\uparrow v_0}$  be a game, and let  $\mathcal{M}$  be a Mealy machine for Leader in  $\mathcal{G}$ . Their product game is the game  $\mathcal{G}_{\uparrow v_0} \otimes \mathcal{M} = (\Pi \cup \{\mathbb{D}\}, V', (V'_i)_i, E', \mu')$  where the player  $\mathbb{D}$ , called Demon, chooses how the machine  $\mathcal{M}$  will run. Formally:*

- $V' = (V \times Q) \cup (V \times Q \times Q)$ ;
- $V'_{\mathbb{L}} = \emptyset$ ,  $V'_i = V_i \times Q \times Q$  for every  $i \in \Pi \setminus \{\mathbb{L}\}$ , and  $V'_{\mathbb{D}} = (V \times Q) \cup (V_{\mathbb{L}} \times Q \times Q)$ ;
- the set  $E'$  contains:
  - the edge  $(u, p)(u, p, q)$  for each  $(p, u, q) \in \Delta$  (if  $u \notin V_{\mathbb{L}}$ ), or  $(p, u, q, v) \in \Delta$  (if  $u \in V_{\mathbb{L}}$ );
  - the edge  $(u, p, q)(v, q)$  for each  $(p, u, q, v) \in \Delta$  (if  $u \in V_{\mathbb{L}}$ );
  - the edge  $(u, p, q)(v, q)$  for each  $(p, u, q) \in \Delta$ , and each  $uv \in E$  (if  $u \notin V_{\mathbb{L}}$ );
- each payoff function  $\mu'_i$  maps every play  $(\pi_0, q_0)(\pi_0, q_0, q_1)(\pi_1, q_1) \dots$  to the payoff  $\mu_i(\pi_0 \pi_1 \dots)$  if  $i \neq \mathbb{D}$ , and to the payoff 0 if  $i = \mathbb{D}$ .

Figure 3 depicts the game  $\mathcal{G}_{\uparrow v_0} \otimes \mathcal{M}$ , when  $\mathcal{G}_{\uparrow v_0}$  is the game of Figure 1 and  $\mathcal{M}$  the machine of Figure 2. Leader is then assimilated to player  $\square$ , and Demon's vertices are represented by dotted boxes. The unreachable vertices have been omitted, and we have given only the non-zero rewards. Since, from the vertex  $(a, q_0, q_1)$ , player  $\circ$  has always the possibility to go to the vertex  $(b, q_1)$  and to get the payoff 1, it can be shown that every NE and every SPE in that game gives player  $\square$  the payoff 1. As we will see now, that means that the strategies compatible with the machine  $\mathcal{M}$  guarantee the payoff 1 to player  $\square$  against Nash-rational or subgame-perfect rational responses, i.e. that  $\mathcal{G}_{\uparrow v_0}, 1 - \varepsilon$ , and  $\mathcal{M}$ , for every  $\varepsilon > 0$ , form a positive instance of the Nash and subgame-perfect RV problems.

► **Theorem 6.** *Let  $\rho \in \{\text{Nash, subgame-perfect}\}$ . Let  $\mathcal{G}_{\uparrow v_0}$  be a game, let  $\mathcal{M}$  be a Mealy machine for Leader in  $\mathcal{G}$ , and let  $t \in \mathbb{Q}$ . Then, every  $\rho$ -response  $\bar{\sigma}_{-\mathbb{L}}$  to every strategy  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$  satisfies  $\mu_{\mathbb{L}}(\langle \bar{\sigma} \rangle) > t$  if and only if every  $\rho$ -equilibrium  $\bar{\tau}$  in the game  $\mathcal{G}_{\uparrow v_0} \otimes \mathcal{M}$  satisfies  $\mu'_{\mathbb{L}}(\langle \bar{\tau} \rangle) > t$ .*

Thus, solving the  $\rho$ -RV problem in the game  $\mathcal{G}_{\uparrow v_0}$  amounts to solve the  $\rho$ -universal threshold problem ( $\rho$ -UT problem) in  $\mathcal{G}_{\uparrow v_0} \otimes \mathcal{M}$ .

► **Problem 7** ( $\rho$ -universal threshold problem in the class  $\mathcal{C}$ ). *Given a game  $\mathcal{G}_{\uparrow v_0} \in \mathcal{C}$ , a player  $i \in \Pi$ , and a threshold  $t \in \mathbb{Q}$ , is every  $\rho$ -equilibrium  $\bar{\sigma}$  in  $\mathcal{G}_{\uparrow v_0}$  such that  $\mu_i(\langle \bar{\sigma} \rangle) > t$ ?*

Moreover, the size of the product game is bounded by a polynomial function of  $\|\mathcal{G}\|$  and  $\|\mathcal{M}\|$ ; and when the game  $\mathcal{G}$  belongs to a class  $\mathcal{C}$  among the three classes defined in Section 2, then all product games constructed from it also belong to  $\mathcal{C}$ . Hence the following.

► **Corollary 8.** *Let  $\mathcal{C}$  be a game class among energy games, DS games, and MP games. Then, in the class  $\mathcal{C}$ , for a given  $\rho \in \{\text{Nash, subgame-perfect}\}$ , the  $\rho$ -UT problem, the  $\rho$ -RV problem, and the deterministic  $\rho$ -RV problem are reducible to each other in polynomial time.*

**Proof.**

■ *The deterministic  $\rho$ -RV problem reduces to the  $\rho$ -RV problem, because a non-deterministic Mealy machine is a Mealy machine.*

■ *The  $\rho$ -UT problem reduces to the deterministic  $\rho$ -RV problem.*

Let  $\mathcal{G}_{\uparrow v_0}$ ,  $i$  and  $t$  form an instance of the  $\rho$ -UT problem. We define the game  $\mathcal{G}'_{\uparrow v_0}$  as equal to the game  $\mathcal{G}_{\uparrow v_0}$ , where Leader has been added to the player set, but controls no vertex. We define  $\mu_{\mathbb{L}} = \mu_i$ . If  $\mathcal{G}$  belongs to the class  $\mathcal{C}$ , so does  $\mathcal{G}'$ . Let  $\mathcal{M}$  be the one-state deterministic Mealy machine on  $\mathcal{G}'$  that never outputs anything. Then, a strategy profile  $\bar{\sigma}$  in  $\mathcal{G}'_{\uparrow v_0}$  is an  $\mathbb{L}$ -fixed  $\rho$ -equilibrium, if and only if it is an  $\mathbb{L}$ -fixed  $\rho$ -equilibrium with  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$ , if and only if the strategy profile  $\bar{\sigma}_{-\mathbb{L}}$  is a  $\rho$ -equilibrium in the game  $\mathcal{G}_{\uparrow v_0}$ . As a consequence  $\mathcal{G}_{\uparrow v_0}$ ,  $i$ , and  $t$  form a positive instance of the  $\rho$ -UT problem, if and only if  $\mathcal{G}'_{\uparrow v_0}$ ,  $\mathcal{M}$ , and  $t$  form a positive instance of the deterministic  $\rho$ -RV problem. Moreover, the latter can be constructed from the former in polynomial time.

■ *The  $\rho$ -RV problem reduces to the  $\rho$ -UT problem, by Th. 6, and since the product game  $\mathcal{G}_{\uparrow v_0} \otimes \mathcal{M}$  can be constructed from  $\mathcal{G}_{\uparrow v_0}$  and  $\mathcal{M}$  in polynomial time. ◀*

## 4 Energy games

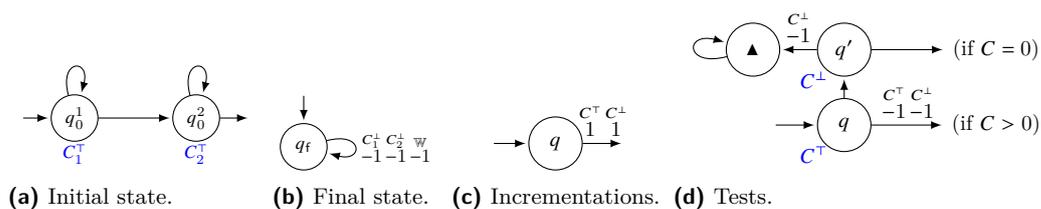
Let us now apply that result to our game classes: first, energy objectives.

**Nash rational verification.** RV problems are undecidable in this class, as we will show by reduction from the halting problem of two-counter machines (the reader who is not familiar with those machines may refer to [7]). However, Nash RV is co-recursively enumerable.

► **Theorem 9.** *In energy games, the Nash RV problem, deterministic or not, is undecidable and co-recursively enumerable.*

**Proof sketch.** We prove here that the Nash UT problem is undecidable and co-recursively enumerable. The theorem will follow by Cor. 8.

■ *Undecidability.* We show undecidability by reduction from the halting problem of a two-counter machine. Let  $\mathcal{K}$  be a two-counter machine. We define an energy game  $\mathcal{G}_{\uparrow q_0^1}$  with five players – players  $C_1^T$ ,  $C_1^+$ ,  $C_2^T$ ,  $C_2^+$ , and  $\mathbb{W}$ , called *Witness* – by assembling the



■ **Figure 4** Gadgets.

gadgets presented in Figure 4 – the rewards that are not presented are equal to 0, and the players controlling relevant vertices are written in blue. Then, a play in  $\mathcal{G}_{\uparrow v_0}$  that does not reach the vertex  $\blacktriangle$  simulates a sequence of transitions of  $\mathcal{K}$ , that can be a valid run or not: at each step, the counter  $C_i$  is captured by the energy level of player  $C_i^\top$ , always equal to the energy level of player  $C_i^\perp$ . For each counter  $C_i$ , the player  $C_i^\perp$  will have a profitable deviation if that play fakes a test to 0, by going to the vertex  $\blacktriangle$ ; and the player  $C_i^\top$  will lose, and therefore have a profitable deviation by staying in  $q_0^i$  if it fakes a positive test. Thus, as shown in the complete version of this proof, every NE outcome in the game  $\mathcal{G}_{\uparrow q_0^1}$  is won by Witness if and only if the machine  $\mathcal{K}$  does not terminate. As a consequence, the halting problem of two-counter machines reduces to the Nash UT problem in energy games, which is therefore undecidable.

- *Co-recursive enumerability.* As shown in the complete version of this proof, in an energy game  $\mathcal{G}_{\uparrow v_0}$ , if there exists an NE that makes some player  $i$  lose, then there exists a finite-memory one. Thus, a semi-algorithm that recognizes the negative instances of the UT problem consists in enumerating the finite-memory complete strategy profiles on  $\mathcal{G}_{\uparrow v_0}$ , and for each of them, to check (by diagonalization):
  - whether it is an NE: that is decidable (in polynomial time), by [7];
  - whether it makes player  $i$  lose: that is recursively enumerable, by constructing step by step its outcome and computing the energy levels on the fly.

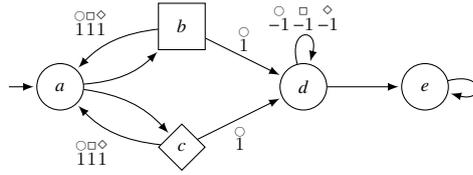
We have a negative instance of the UT problem if and only if at least one finite-memory strategy profile satisfies those two conditions. The Nash UT problem is therefore co-recursively enumerable.  $\blacktriangleleft$

**Subgame-perfect rational verification.** In the subgame-perfect setting, the previous construction could also prove undecidability. But we choose to present a refinement of it, that proves a stronger result.

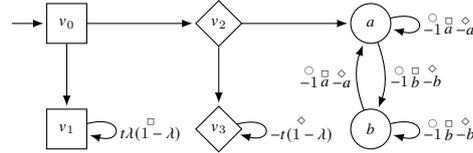
► **Theorem 10.** *In energy games, the subgame-perfect RV problem, deterministic or not, is undecidable, even when Leader plays against only two players.*

Again, the proof shows that, in particular, that problem is not recursively enumerable in energy games. It might still be the case that it is co-recursively enumerable. That would in particular be the case if finite memory was sufficient for an SPE to make any player  $i$  lose, when that is possible, as in the case of NEs. Unfortunately, one cannot follow this approach, because that statement is false: in order to be able to punish some player, without making another player lose, an SPE may have to memorize their energy levels, and therefore require infinite memory, as it will be the case in the example that follows. We leave therefore the question open.

► **Proposition 11.** *In the energy game presented in Figure 5, there exists an SPE that makes player  $\square$  lose, but no finite memory SPE can achieve that result.*



■ **Figure 5** A game where infinite memory is necessary to make player  $\square$  lose.



■ **Figure 6** A game constructed from an instance of TDS.

### 5 Discounted-sum games

We will now move to DS objectives. First, let us define the following decision problem.

► **Problem 12** (Target discounted-sum problem). *Given four quantities  $\lambda, a, b, t \in \mathbb{Q}$  with  $0 < \lambda < 1$ , is there a sequence  $(u_n)_{n \in \mathbb{N}} \in \{a, b\}^\omega$  such that  $\sum_{n \in \mathbb{N}} u_n \lambda^n = t$ ?*

Although it is a quite natural problem that appears in many different fields, the target discounted-sum (TDS) problem turns out to be surprisingly hard to solve, and its decidability status is still open. The interested reader may refer to [1] for more details. The following theorem shows that RV problems are at least as difficult.

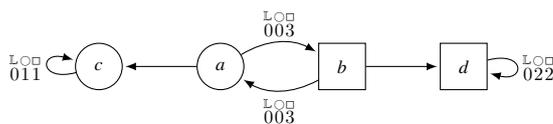
► **Theorem 13.** *The TDS problem reduces to the complements of the (deterministic) Nash rational and subgame-perfect RV problems in discounted-sum games.*

**Proof.** We present here a reduction to the complements of the Nash universal and subgame-perfect UT problems; the result follows by Cor. 8. Let  $a, b, t \in \mathbb{Q}$ , let  $\lambda \in \mathbb{Q} \cap (0, 1)$ , and let  $\mathcal{G}_{\uparrow v_0}$  be the DS game of Figure 6, with discount factor  $\lambda$ . In that game, there exists an NE  $\bar{\sigma}$  with  $\mu_\circ(\langle \bar{\sigma} \rangle) < 0$ , if and only if there exists an SPE  $\bar{\sigma}$  with  $\mu_\circ(\langle \bar{\sigma} \rangle) < 0$ , if and only if  $a, b, t$ , and  $\lambda$  form a positive instance of the TDS problem.

Indeed, if such an NE or SPE exists, it necessarily reaches the vertex  $a$ . But then, player  $\square$  must get at least the payoff  $\mu_\square(v_0 v_1^\omega) = t\lambda^2$ , and player  $\diamond$  the payoff  $\mu_\diamond(v_0 v_2 v_3^\omega) = -t\lambda^2$ , otherwise they would have a profitable deviation. If such a play exists, then we have a positive instance of the TDS problem. Conversely, from such a positive instance, one can construct a play from  $v_0$  in which player  $\circ$  gets the payoff  $\frac{\lambda^2}{1-\lambda}$ , player  $\square$  the payoff  $t\lambda^2$ , and player  $\diamond$  the payoff  $-t\lambda^2$ , and none of them has a profitable deviation in any subgame. ◀

The previous theorem suggests that finding algorithms solving those problems is a very ambitious objective. However, in the sequel, we will show that like the TDS problem, the RV problems are recursively enumerable. The key idea is the following: a property of DS objectives is that when a play gives to some player a payoff that is strictly smaller than some threshold, that can be seen on finite prefixes of those plays. Therefore, although strategy profiles are in general infinite objects that exist in uncountable number, profitable deviations can be found by analyzing their behaviors on a finite (but unbounded) number of histories.

► **Theorem 14.** *In DS games, the Nash rational and the subgame-perfect RV problems, deterministic or not, are recursively enumerable.*



■ **Figure 7** The temptation of chaos: an illustration.

## 6 Mean-payoff games

**Classical rational verification.** Let us now end with MP games. The reduction from RV problems to UT problems enables us to apply results and methods that already exist in the literature.

► **Theorem 15.** *In the class of MP games, the Nash rational and the subgame-perfect RV problems, deterministic or not, are coNP-complete.*

**The temptation of chaos.** It is now worth noting that the definition we gave of RV entails, in the case of MP games, results that may be considered as counter-intuitive. For instance, consider the game of Figure 7, where Leader owns no vertex, and consider the only (vacuous) strategy available for Leader. Does that strategy guarantee a payoff greater than 1? Intuitively, it does not, since Leader always receives the payoff 0. But still, that strategy, that game, and that threshold form a positive instance of subgame-perfect RV, because no  $\mathbb{L}$ -fixed SPE exists in that game (see [4]). More generally, the definition we give of RV considers that a *good* strategy for Leader is a strategy such that for every response of the environment that is rational, the generated outcome observes some specification. But a strategy is then good, in that sense, if *no* rational response of the environment exists: that is the phenomenon that we can call *temptation of chaos*. While that case does never occur in energy and DS games, where rational responses are always guaranteed to exist (as we will see below), it must be considered in MP games.

**Achaotic rational verification.** To avoid such phenomena, we introduce an alternative definition of RV, *achaotic RV*: a good strategy for Leader will be a strategy that guarantees the given threshold against every response that is *as rational as possible*. To define that problem, we need quantitative relaxations to the notions of NEs and SPEs. Let  $\mathcal{G}_{\uparrow v_0}$  be a game. Let  $\varepsilon \geq 0$ . The strategy profile  $\bar{\sigma}$  is an  $\varepsilon$ -NE (resp.  $\mathbb{L}$ -fixed  $\varepsilon$ -NE) in  $\mathcal{G}_{\uparrow v_0}$  if and only if for each  $i \in \Pi$  (resp.  $\Pi \setminus \{\mathbb{L}\}$ ) and every deviation  $\sigma'_i$  of  $\sigma_i$ , the inequality  $\mu_i(\langle \sigma'_i, \bar{\sigma}_{-i} \rangle) \leq \mu_i(\langle \bar{\sigma} \rangle) + \varepsilon$  holds: no deviation is profitable *by more than*  $\varepsilon$ . Note that 0-NEs coincide with NEs. We derive from that notion, as expected, the notions of ( $\mathbb{L}$ -fixed)  $\varepsilon$ -SPEs,  $\varepsilon$ -Nash and  $\varepsilon$ -subgame-perfect responses, and the notations  $\varepsilon\text{NR}(\sigma_{\mathbb{L}})$ ,  $\varepsilon\text{SPR}(\sigma_{\mathbb{L}})$ , and  $\varepsilon\rho\text{R}(\sigma_{\mathbb{L}})$ . We can now define our decision problem.

► **Problem 16** (Achaotic (deterministic)  $\rho$ -RV in the class  $\mathcal{C}$ ). *Given a game  $\mathcal{G}_{\uparrow v_0} \in \mathcal{C}$ , a threshold  $t \in \mathbb{Q}$ , and a Mealy machine (resp. a deterministic Mealy machine)  $\mathcal{M}$  on  $\mathcal{G}$ , does there exist  $\varepsilon \geq 0$  satisfying:*

- $\varepsilon\rho\text{R}(\sigma_{\mathbb{L}}) \neq \emptyset$  for some strategy  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$ ;
- and  $\mu_{\mathbb{L}}(\langle \sigma_{\mathbb{L}}, \bar{\sigma}_{-\mathbb{L}} \rangle) > t$  for every  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$ , and every  $\bar{\sigma}_{-\mathbb{L}} \in \varepsilon\rho\text{R}(\sigma_{\mathbb{L}})$ ?

We will prove below that in mean-payoff games, there exists a least quantity  $\varepsilon_{\min}$  such that  $\varepsilon_{\min}\rho$ -responses to a given strategy  $\sigma_{\mathbb{L}}$  exist. For instance, in the example depicted by Figure 7, we have  $\varepsilon_{\min} = 1$ . Thus, we can rephrase the achaotic RV problems as follows: given a game  $\mathcal{G}_{\uparrow v_0}$ , a threshold  $t \in \mathbb{Q}$  and a Mealy machine  $\mathcal{M}$ , do we have  $\mu_{\mathbb{L}}(\langle \sigma_{\mathbb{L}}, \bar{\sigma}_{-\mathbb{L}} \rangle) > t$  for every  $\sigma_{\mathbb{L}} \in \text{Comp}_{\uparrow v_0}(\mathcal{M})$  and every  $\bar{\sigma}_{-\mathbb{L}} \in \varepsilon_{\min}\rho\text{R}(\sigma_{\mathbb{L}})$ ?

Among the problems we study here, this new definition is relevant in only one case: subgame-perfect RV in MP games. In all other cases, the RV problems are equivalent to their achaotic versions, because Nash and subgame-perfect responses are guaranteed to exist.

► **Proposition 17.** *Let  $C$  be a class of games, among the classes of energy games and DS games. Let  $\rho \in \{\text{Nash}, \text{subgame-perfect}\}$ . Then, the positive instances of the achaotic  $\rho$ -RV problem in  $C$  are exactly the positive instances of the  $\rho$ -RV problem. Similarly, the positive instances of the achaotic Nash-RV problem in MP games are exactly the positive instances of the  $\rho$ -RV problem.*

Now, an optimal algorithm for that problem in MP games requires the following lemma: in each game, there exists a least  $\varepsilon$  such that  $\varepsilon$ -SPEs exist, and it can be written with a polynomially bounded number of bits. To prove that, we need to use the notion of *negotiation function*, defined in [4]: a function from vertex labellings to vertex labellings whose least  $\varepsilon$ -fixed point (i.e., the least vertex labelling  $\lambda$  that is a fixed point of that function up to  $\varepsilon$ ) characterizes  $\varepsilon$ -SPEs. Our result can be obtained by revisiting a proof of [5], that was designed to bound the number of bits required to write that least  $\varepsilon$ -fixed point, for a fixed  $\varepsilon$ . Hereafter, we write  $\|\varepsilon\|$  for the number of bits required to write  $\varepsilon$  in a usual encoding.

► **Lemma 18.** *There exists a polynomial  $P_1$  such that in every mean-payoff game  $\mathcal{G}_{\uparrow v_0}$ , there exists  $\varepsilon_{\min}$  with  $\|\varepsilon_{\min}\| \leq P_1(\|\mathcal{G}\|)$  such that  $\varepsilon_{\min}$ -SPEs exist in  $\mathcal{G}_{\uparrow v_0}$ , and  $\varepsilon$ -SPEs, for every  $\varepsilon < \varepsilon_{\min}$ , do not.*

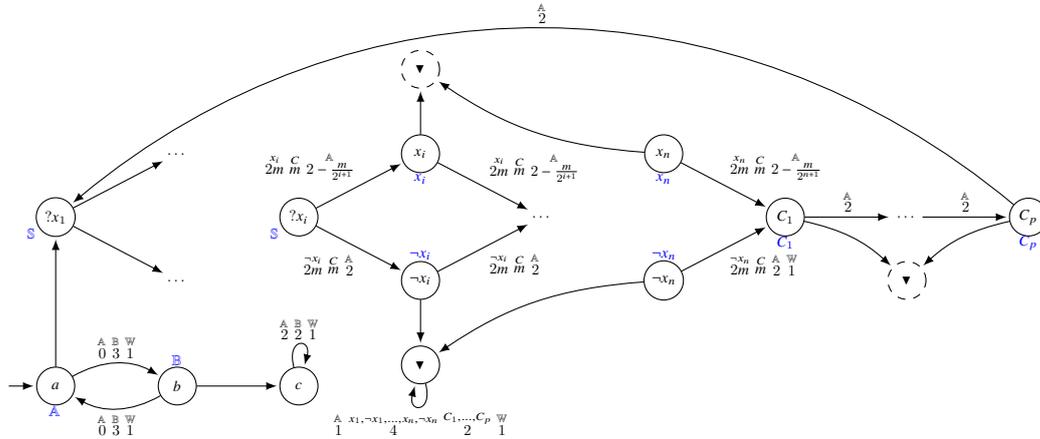
We are now equipped to prove the following theorem.

► **Theorem 19.** *In the class of mean-payoff games, the achaotic subgame-perfect RV problem, deterministic or not, is  $\text{P}^{\text{NP}}$ -complete.*

**Proof sketch.** Using Lem. 18 and the same arguments as in the proof of Th. 6, those two problems are interreducible with the following one: given  $\mathcal{G}_{\uparrow v_0}$  and a  $t \in \mathbb{Q}$ , does every  $\varepsilon_{\min}$ -SPE  $\bar{\sigma}$  in  $\mathcal{G}_{\uparrow v_0}$  satisfy  $\mu_{\mathbb{L}}(\langle \bar{\sigma} \rangle) > t$ ? Let us prove  $\text{P}^{\text{NP}}$ -completeness for that problem.

- *Easiness.* By [5], there is an NP algorithm deciding, given  $\varepsilon$  and  $\mathcal{G}_{\uparrow v_0}$ , whether there is an  $\varepsilon$ -SPE in  $\mathcal{G}_{\uparrow v_0}$ , i.e. whether  $\varepsilon \geq \varepsilon_{\min}$ . Using Lem. 18 and the inequality  $\varepsilon_{\min} \leq 2 \max_{i,uv} |r_i(uv)|$ , a dichotomous search can thus compute  $\varepsilon_{\min}$  using polynomially many calls to that algorithm. Then, one last call can decide whether there exists an  $\varepsilon_{\min}$ -SPE  $\bar{\sigma}$  such that  $\mu_i(\langle \bar{\sigma} \rangle) \leq t$ .
- *Hardness.* We proceed by reduction from the following  $\text{P}^{\text{NP}}$ -complete problem: given a Boolean formula  $\varphi$  in conjunctive normal form over the ordered variables  $x_1, \dots, x_n$ , is the lexicographically first valuation  $v_{\min}$  satisfying  $\varphi$  such that  $v_{\min}(x_n) = 1$ ? (and in particular, does such a valuation exist?) Let us write  $\varphi = \bigwedge_{j=1}^p C_j$ . We construct a game  $\mathcal{G}_{\uparrow a}$ , with a player called *Witness* and written  $\mathbb{W}$ , in which there exists an  $\varepsilon_{\min}$ -SPE  $\bar{\sigma}$  such that  $\mu_{\mathbb{W}}(\langle \bar{\sigma} \rangle) \leq 0$  if and only if  $\varphi$  is satisfiable and  $v_{\min}(x_n) = 1$ . That game, depicted in Figure 8 (unmentioned rewards are equal to 0, and we write  $m = 2n + p$ ), has  $2n + p + 4$  players: the literal players  $x_1, \neg x_1, \dots, x_n, \neg x_n$ ; the clause players  $C_1, \dots, C_p$ ; the player *Solver*, written  $\mathbb{S}$ ; the player *Witness*, written  $\mathbb{W}$ ; the player *Alice*, written  $\mathbb{A}$ ; and the player *Bob*, written  $\mathbb{B}$ .

This game is based on the classical example of MP game in which SPEs do not exist, already presented in Section 6. In the latter, from the vertex  $a$ , Alice can access a sink vertex, where Bob and her both get the payoff 1. Here, they access instead to a region where the choices of Solver define a valuation of  $x_1, \dots, x_n$  – unless one of the literal players chooses to go to the sink vertex  $\blacktriangledown$ , which will be a profitable deviation if Solver makes inconsistent choices (one literal and, later, its negation). That valuation  $v$



■ **Figure 8** The game  $\mathcal{G}_{\uparrow a}$ .

defines Alice's payoff  $\mu_A(\pi) = 2 - \sum_{i=1}^n \frac{v(x_i)}{2^i}$ , and therefore defines how much deviating and reaching  $c$  is profitable for her. Consequently, as we show in the complete version of this proof, the valuation  $v_{\min}$  is the binary encoding of the quantity  $\varepsilon_{\min}$ , and there is an  $\varepsilon_{\min}$ -SPE in which Witness gets the payoff 0 or less if and only if  $v_{\min}(x_n) = 1$ . ◀

## References

- 1 Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *30th Annual ACM/IEEE Symp. on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 750–761. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.74.
- 2 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jiri Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems, 6th Int. Conf., FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proc.*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008. doi:10.1007/978-3-540-85778-5\_4.
- 3 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications – 10th Int. Conf., LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proc.*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016.
- 4 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. In Serge Haddad and Daniele Varacca, editors, *32nd Int. Conf. on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.8.
- 5 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. The complexity of SPEs in mean-payoff games. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th Int. Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 116:1–116:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.116.
- 6 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. On the complexity of SPEs in parity games. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conf. on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 10:1–10:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.10.

- 7 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Rational verification and checking for nash and subgame-perfect equilibria in graph games. *CoRR*, abs/2301.12913, 2023. doi:10.48550/arXiv.2301.12913.
- 8 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th Int. Conf. on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.13.
- 9 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In *Developments in Language Theory – 21st Int. Conf., DLT 2017, Liège, Belgium, August 7-11, 2017, Proc.*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017.
- 10 Véronique Bruyère. Synthesis of equilibria in infinite-duration games on graphs. *ACM SIGLOG News*, 8(2):4–29, 2021. doi:10.1145/3467001.3467003.
- 11 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Pareto-rational verification. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd Int. Conf. on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPICs*, pages 33:1–33:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CONCUR.2022.33.
- 12 Krishnendu Chatterjee and Vishwanath Raman. Synthesizing protocols for digital contract signing. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation – 13th Int. Conf., VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proc.*, volume 7148 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2012. doi:10.1007/978-3-642-27940-9\_11.
- 13 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.
- 14 Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors. *The Adversarial Stackelberg Value in Quantitative Games*, volume 168 of *Leibniz Int. Proc. in Informatics (LIPICs)*, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.127.
- 15 Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proc. ACM Program. Lang.*, 1(ICFP):34:1–34:29, 2017. doi:10.1145/3110278.
- 16 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th Int. Conf., TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proc.*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2\_16.
- 17 János Flesch and Arkadi Predtetchinski. A characterization of subgame-perfect equilibrium plays in borel games of perfect information. *Math. Oper. Res.*, 42(4):1162–1179, 2017. doi:10.1287/moor.2016.0843.
- 18 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/j.artint.2020.103353.
- 19 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. On the complexity of rational verification. *CoRR*, abs/2207.02637, 2022. doi:10.48550/arXiv.2207.02637.
- 20 Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR 2001 – Concurrency Theory, 12th Int. Conf., Aalborg, Denmark, August 20-25, 2001, Proc.*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565. Springer, 2001. doi:10.1007/3-540-44685-0\_37.

- 21 Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *J. Comput. Secur.*, 11(3):399–430, 2003. URL: <http://content.iospress.com/articles/journal-of-computer-security/jcs185>, doi:10.3233/jcs-2003-11307.
- 22 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.
- 23 Noémie Meunier. *Multi-Player Quantitative Games: Equilibria and Algorithms*. PhD thesis, Université de Mons, 2016.
- 24 John F. Nash. Equilibrium points in  $n$ -person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.
- 25 Martin J. Osborne. *An introduction to game theory*. Oxford Univ. Press, 2004.
- 26 Dorsa Sadigh, Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. Planning for autonomous cars that leverage effects on human actions. In David Hsu, Nancy M. Amato, Spring Berman, and Sam Ade Jacobs, editors, *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 – June 22, 2016*, 2016. doi:10.15607/RSS.2016.XII.029.
- 27 Thomas Steeples, Julian Gutierrez, and Michael J. Wooldridge. Mean-payoff games with  $\omega$ -regular specifications. In Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé, editors, *AAMAS '21: 20th Int. Conf. on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 1272–1280. ACM, 2021. doi:10.5555/3463952.3464099.
- 28 Michael Ummels. Rational Behaviour and Strategy Construction in Infinite Multiplayer Games. Diploma thesis, RWTH Aachen, 2005. URL: <http://www.logic.rwth-aachen.de/~ummels/diplom.pdf>.
- 29 Michael Ummels. The complexity of nash equilibria in infinite multiplayer games. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th Int. Conf., FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 – April 6, 2008. Proc.*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2008. doi:10.1007/978-3-540-78499-9\_3.
- 30 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015.
- 31 Uri Zwick and Mike Paterson. *The complexity of mean payoff games*, volume 959, pages 1–10. Springer, April 2006. doi:10.1007/BFb0030814.



# On Property Testing of the Binary Rank

Nader H. Bshouty ✉

Technion, Haifa, Israel

---

## Abstract

---

Let  $M$  be an  $n \times m$   $(0, 1)$ -matrix. We define the  $s$ -binary rank, denoted as  $\text{br}_s(M)$ , of  $M$  as the minimum integer  $d$  such that there exist  $d$  monochromatic rectangles covering all the 1-entries in the matrix, with each 1-entry being covered by at most  $s$  rectangles. When  $s = 1$ , this corresponds to the binary rank, denoted as  $\text{br}(M)$ , which is well-known in the literature and has many applications.

Let  $R(M)$  and  $C(M)$  denote the sets of rows and columns of  $M$ , respectively. Using the result of Sgall [10], we establish that if  $M$  has an  $s$ -binary rank at most  $d$ , then  $|R(M)| \cdot |C(M)| \leq \binom{d}{\leq s} 2^d$ , where  $\binom{d}{\leq s} = \sum_{i=0}^s \binom{d}{i}$ . This bound is tight, meaning that there exists a matrix  $M'$  with an  $s$ -binary rank of  $d$ , for which  $|R(M')| \cdot |C(M')| = \binom{d}{\leq s} 2^d$ .

Using this result, we present novel one-sided adaptive and non-adaptive testers for  $(0, 1)$ -matrices with an  $s$ -binary rank at most  $d$  (and exactly  $d$ ). These testers require  $\tilde{O}\left(\binom{d}{\leq s} 2^d / \epsilon\right)$  and  $\tilde{O}\left(\binom{d}{\leq s} 2^d / \epsilon^2\right)$  queries, respectively.

For a fixed  $s$ , this improves upon the query complexity of the tester proposed by Parnas et al. in [9] by a factor of  $\tilde{\Theta}(2^d)$ .

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** Property testing, binary rank, Boolean rank

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.27

## 1 Introduction

Let  $M$  be an  $n \times m$   $(0, 1)$ -matrix. We define the  $s$ -binary rank, denoted as  $\text{br}_s(M)$ , of  $M$  as the minimum integer  $d$  such that there exist  $d$  sets (rectangles)  $I_k \times J_k$ , where  $I_k \subseteq [n] := \{1, \dots, n\}$  and  $J_k \subseteq [m]$  for  $k \in [d]$  that satisfy the conditions: for every  $(i, j) \in [n] \times [m]$  where  $M[i, j] = 1$ , there is at least one and at most  $s$  integer  $t \in [d]$  such that  $(i, j) \in I_t \times J_t$  (each 1-entry in  $M$  is covered by at least one and at most  $s$  monochromatic rectangles). Additionally,  $M[i, j] = 1$  for all  $(i, j) \in I_k \times J_k$  for  $k \in [d]$  (monochromatic rectangles).

When  $s = 1$ , the  $s$ -binary rank  $\text{br}_1(M)$  is known as the binary rank, denoted as  $\text{br}(M)$ . When  $s = \infty$ , the  $s$ -binary rank  $\text{br}_\infty(M)$  is referred to as the Boolean rank. Both of these concepts are well-known in the literature. You can explore many applications of these concepts by referring to notable sources such as Amilhastre and Vigneron [1], Chalermsook et al. [3], and Gregory et al. [5]. These references, along with the internal citations, provide an extensive collection of related works with additional applications.

The binary rank can also be defined as follows: the binary rank of an  $n \times m$   $(0, 1)$ -matrix  $M$  is equal to the minimum  $d$  such that there exist an  $n \times d$   $(0, 1)$ -matrix  $N$  and a  $d \times m$   $(0, 1)$ -matrix  $L$  satisfying  $M = NL$ . The binary rank can also be interpreted as the minimum number of bipartite cliques required to partition all the edges of a bipartite graph with adjacency matrix  $M$ . Similarly, the  $s$ -binary rank of  $M$  is the minimum number of bipartite cliques needed to cover all the edges of a bipartite graph with adjacency matrix  $M$ , with each edge being covered by at most  $s$  bipartite cliques. In [3], Chalermsook et al. show that approximating the binary rank within a factor of  $n^{1-\delta}$  for any given  $\delta$  is NP-hard.

A *property-testing* algorithm, also known as a *tester*, for the  $s$ -binary rank [9], takes as input  $0 < \epsilon < 1$ , integers  $d$ ,  $n$ , and  $m$ , and has query access to the entries of an  $n \times m$   $(0, 1)$ -matrix  $M$ . If  $M$  has an  $s$ -binary rank at most  $d$  (or exactly  $d$ ), the tester accepts with



© Nader H. Bshouty;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 27; pp. 27:1–27:14

Leibniz International Proceedings in Informatics



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

probability at least  $2/3$ . If  $M$  is  $\epsilon$ -far from having an  $s$ -binary rank at most  $d$  (or exactly  $d$ ), meaning that modifying more than an  $\epsilon$ -fraction of the entries of  $M$  is required to obtain a matrix with an  $s$ -binary rank at most  $d$  (or exactly  $d$ ), then the tester rejects with probability at least  $2/3$ . If the tester accepts matrices with an  $s$ -binary rank at most  $d$  (or exactly  $d$ ) with probability 1, it is referred to as a *one-sided error tester*. In *adaptive testing*, the queries can depend on the answers to previous queries, while in *non-adaptive testing*, all queries are predetermined by the tester in advance. The objective is to construct a tester that makes the fewest number of queries possible.

The testability of the  $s$ -binary rank at most  $d$  of  $(0,1)$ -matrices was studied in [8, 9]. In [8], Nakar and Ron presented a non-adaptive one-sided error tester for  $s = 1$  that makes  $\tilde{O}(2^{4d}/\epsilon^4)$  queries. In [9], Parnas et al. gave non-adaptive and adaptive one-sided error testers for  $s = 1$  that makes  $O(2^{2d}/\epsilon^2)$  and  $O(2^{2d}/\epsilon)$  queries, respectively. The results presented in [9] also hold to the  $s$ -binary rank at most  $d$ .

In this paper, we establish the following theorems for the testability of the  $s$ -binary rank at most  $d$  (or exactly  $d$ ):

► **Theorem 1.** *There exists an adaptive one-sided error tester for  $s$ -binary rank of  $n \times m$   $(0,1)$ -matrices that makes  $\tilde{O}\left(\binom{d}{\leq s} 2^d/\epsilon\right)$  queries.*

► **Theorem 2.** *There exists a non-adaptive one-sided error tester for  $s$ -binary rank of  $n \times m$   $(0,1)$ -matrices that makes  $\tilde{O}\left(\binom{d}{\leq s} 2^d/\epsilon^2\right)$  queries.*

For fixed  $s$ , this improves the query complexity of Parnas et al. in [9] by a factor of  $\tilde{O}(2^d)$ .

## 1.1 Our Approach

The tester of Parnas et al. [9] uses the fact that if  $M'$  is a  $k \times k$  sub-matrix of  $M$  and  $M'$  is of  $s$ -binary rank at most  $d$ , then the following properties hold:

1.  $M'$  has at most  $2^d$  distinct rows and at most  $2^d$  distinct columns.
2. If  $M$  is  $\epsilon$ -far from having  $s$ -binary rank at most  $d$ , then extending  $M'$  by one more uniformly at random row and column from  $M$  yields a  $(k+1) \times (k+1)$  sub-matrix  $M''$  of  $M$  that, with probability at least  $\Omega(\epsilon)$ , satisfies: either the number of distinct rows in  $M''$  is greater by one than the number of distinct rows in  $M'$ , or the number of distinct columns in  $M''$  is greater by one than the number of distinct columns in  $M'$ .

So, their adaptive tester starts from an empty matrix  $M' = ()$  and then runs  $O(2^d/\epsilon)$  iterations. At every iteration, if  $M'$  is of size  $(2^d + 1) \times (2^d + 1)$  it rejects. Otherwise, it extends  $M'$  by uniformly at random one row and one column. Let  $M''$  be the resulting sub-matrix. If the  $s$ -binary rank of  $M''$  is greater than  $d$ , the tester rejects. If the number of distinct rows or columns in  $M''$  is greater than the number in  $M'$ , then it continues to the next iteration with  $M' \leftarrow M''$ . Otherwise, it continues to the next iteration with  $M'$ . If, after  $O(2^d/\epsilon)$  iterations,  $M'$  has  $s$ -binary rank at most  $d$ , the tester accepts.

If the  $s$ -binary rank of  $M$  is at most  $d$ , then every sub-matrix of  $M$  has an  $s$ -binary rank at most  $d$ , and the tester accepts. If  $M$  is  $\epsilon$ -far from having  $s$ -binary rank at most  $d$ , then: since, at each iteration, with probability at least  $\Omega(\epsilon)$ , the number of distinct rows or columns of  $M'$  is increased by one, and since matrices of  $s$ -binary rank  $d$  have at most  $2^d$  distinct rows and at most  $2^d$  distinct columns, with high probability, we either obtain  $M'$  with an  $s$ -binary rank greater than  $d$ , or  $M'$  reaches to the dimension of  $(2^d + 1) \times (2^d + 1)$ . In both cases, the tester rejects. The query complexity of the tester is  $O(2^{2d}/\epsilon)$ , which is the worst-case number of the entries of the matrix  $M'$ ,  $O(2^{2d})$ , times the number of trials  $O(1/\epsilon)$  for extending  $M'$  by one row and one column.

We now give our approach. Call a sub-matrix  $M'$  of  $M$  *perfect* if it has distinct rows and distinct columns. Our adaptive tester uses the fact that if  $M'$  is a perfect  $k \times k'$  sub-matrix of  $M$  of  $s$ -binary rank  $d$ , then

1.  $kk' \leq \binom{d}{\leq s} 2^d$ .
2. If  $M$  is  $\epsilon$ -far from having  $s$ -binary rank at most  $d$ , then at least one of the following occurs<sup>1</sup>
  - a. With probability at least  $\Omega(\epsilon)$ , extending  $M'$  by one uniformly at random column of  $M$ , gives a perfect  $k \times (k' + 1)$  sub-matrix  $M''$  of  $M$ .
  - b. With probability at least  $\Omega(\epsilon)$ , extending  $M'$  by one uniformly at random row of  $M$ , gives a perfect  $(k + 1) \times k'$  sub-matrix  $M''$  of  $M$ .
  - c. With probability at least  $\Omega(\epsilon)$ , extending  $M'$  by one uniformly at random column and one uniformly at random row of  $M$ , gives a perfect<sup>2</sup>  $(k + 1) \times (k' + 1)$  sub-matrix  $M''$  of  $M$ .

Item 1 follows from Sgall's result in [10] (See Section 3), and item 2 is Claim 10 in Parnas et al [9]. Now, the tester's strategy is as follows. If  $k \leq k'$ , the tester first tries to extend  $M'$  with a new column. If it succeeds, it moves to the next iteration. Otherwise, it tries to extend  $M'$  with a new row. If it succeeds, it moves to the next iteration. Otherwise, it tries to extend  $M'$  with a new row and a new column. If it succeeds, it moves to the next iteration. If it fails, it accepts. If  $k' < k$ , it starts with the row, then the column, and then both.

Using this strategy, we show that the query complexity will be, at most, the order of the size  $kk' \leq \binom{d}{\leq s} 2^d$  of  $M'$  times the number of trials,  $\tilde{O}(1/\epsilon)$ , to find the new row, column, or both. This achieves the query complexity in Theorem 1.

For the non-adaptive tester, the tester, uniformly at random, chooses  $t = \tilde{O}\left(\binom{d}{\leq s} 2^d / \epsilon^2\right)$  rows  $r_1, \dots, r_t \in [n]$  and  $t$  columns  $c_1, \dots, c_t \in [m]$  and queries all  $M[r_i, c_j]$  for all  $i \cdot j \leq t$  and puts them in a table. Then it runs the above non-adaptive tester. When the non-adaptive tester asks for uniformly at random row or column, it provides the next element  $r_i$  or  $c_j$ , respectively. The queries are then answered from the table. We show that the adaptive algorithm does not need to make queries that are not in the table before it halts. This achieves the query complexity in Theorem 2.

## 1.2 Other Rank Problems

The *real rank* of a  $n \times m$ -matrix  $M$  over any field  $F$  is the minimal  $d$ , such that there is a  $n \times d$  matrix  $N$  over  $F$  and a  $d \times m$  matrix  $L$  over  $F$  such that  $M = NL$ . The testability of the real rank was studied in [2, 6, 7]. In [2], Balcan et al. gave a non-adaptive tester for the real rank that makes  $\tilde{O}(d^2/\epsilon)$  queries. They also show that this query complexity is optimal.

The Boolean rank ( $\infty$ -binary rank) was studied in [8, 9]. Parnas et al. in [9] gave a non-adaptive tester for the Boolean rank that makes  $\tilde{O}(d^4/\epsilon^4)$  queries<sup>3</sup>.

<sup>1</sup> Note that what we have in a-c is not precisely what we use in the algorithm and its proof of correctness. For the exact statement, please refer to Claim 13. It can be observed that both statements are equivalent, allowing for a change in the constant within the  $\Omega$  notation.

<sup>2</sup> It may happen that events (a) and (b) do not occur and (c) does

<sup>3</sup> The query complexity in [9] is  $\tilde{O}(d^4/\epsilon^6)$ . We've noticed that Lemma 3 in [9] is also true when we replace  $(\epsilon^2/64)n^2$  with  $(\epsilon/4)n^2$ . To prove that, in the proof of Lemma 3, replace Modification rules 1 and 2 with the following modification: Modify to 0 all beneficial entries. This gives the result stated here, [4].

## 2 Definitions and Preliminary Results

Let  $M$  be a  $n \times m$   $(0, 1)$ -matrix. We denote by  $R(M)$  and  $C(M)$  the set of rows and columns of  $M$ , respectively. The number of distinct rows and columns of  $M$  are denoted by  $r(M) = |R(M)|$  and  $c(M) = |C(M)|$ , respectively. The *binary rank* of a  $n \times m$ -matrix  $M$ ,  $\text{br}(M)$ , is equal to the minimal  $d$ , where there is a  $n \times d$   $(0, 1)$ -matrix  $N$  and a  $d \times m$   $(0, 1)$ -matrix  $L$  such that  $M = NL$ .

We define the  $s$ -binary rank,  $\text{br}_s(M)$ , of  $M$  to be the minimal integer  $d$  such that there are  $d$  sets (rectangles)  $I_k \times J_k$  where  $I_k \subseteq [n] := \{1, \dots, n\}$ ,  $J_k \subseteq [m]$ ,  $k \in [d]$  such that  $M[i, j] = 1$  for all  $(i, j) \in I_k \times J_k$ ,  $k \in [d]$  (monochromatic rectangles) and for every  $(i, j) \in [n] \times [m]$  where  $M[i, j] = 1$  there are at least one and at most  $s$  integers  $t \in [d]$  such that  $(i, j) \in I_t \times J_t$  (each 1-entry in  $M$  is covered by at least one and at most  $s$  monochromatic rectangles).

We now prove.

► **Lemma 3.** *Let  $M$  be a  $n \times m$   $(0, 1)$ -matrix. The  $s$ -binary rank of  $M$ ,  $\text{br}_s(M)$ , is equal to the minimal integer  $d$ , where there is a  $n \times d$   $(0, 1)$ -matrix  $N$  and a  $d \times m$   $(0, 1)$ -matrix  $L$  such that: For  $P = NL$ ,*

1. *For every  $(i, j) \in [n] \times [m]$ ,  $M[i, j] = 0$  if and only if  $P[i, j] = 0$ .*
2. *For every  $(i, j) \in [n] \times [m]$ ,  $P[i, j] \leq s$ .*

**Proof.** If  $M$  is of  $s$ -binary rank  $d$ , then there are rectangles  $\{I_k \times J_k\}_{k \in [d]}$ ,  $I_k \subseteq [n]$ ,  $J_k \subseteq [m]$ ,  $k \in [d]$  such that  $M[i, j] = 1$  for all  $(i, j) \in I_k \times J_k$ ,  $k \in [d]$  and for every  $(i, j) \in [n] \times [m]$  where  $M[i, j] = 1$  there are at least one and at most  $s$  integers  $t \in [d]$  such that  $(i, j) \in I_t \times J_t$ . Define row vectors  $a^{(k)} \in \{0, 1\}^n$  and  $b^{(k)} \in \{0, 1\}^m$  where  $a_i^{(k)} = 1$  iff (if and only if)  $i \in I_k$ , and  $b_j^{(k)} = 1$  iff  $j \in J_k$ . Then define<sup>4</sup>  $P = a^{(1)'}b^{(1)} + \dots + a^{(d)'}b^{(d)}$ . It is easy to see that  $(a^{(k)'}b^{(k)})[i, j] = 1$  iff  $(i, j) \in I_k \times J_k$ . Therefore,  $P[i, j] = 0$  iff  $M[i, j] = 0$  and  $P[i, j] \leq s$  for all  $(i, j) \in [n] \times [m]$ . Define the  $n \times d$  matrix  $N = [a^{(1)'} | \dots | a^{(d)'}]$  and the  $d \times m$  matrix  $L = [b^{(1)} | \dots | b^{(d)}]'$ . It is again easy to see that  $P = NL$ .

The other direction can be easily seen by tracing backward in the above proof. ◀

We now prove the following,

► **Lemma 4.** *Let  $P$  be a  $n \times m$  matrix. Let  $N$  and  $L$  be  $n \times d$   $(0, 1)$ -matrix and  $d \times m$   $(0, 1)$ -matrix, respectively, such that  $P = NL$ . Then  $r(P) \leq r(N)$  and  $c(P) \leq c(L)$ .*

**Proof.** We prove the result for  $r$ . The proof for  $c$  is similar. Let  $r_1, \dots, r_n$  be the rows of  $N$  and  $p_1, \dots, p_n$  be the rows of  $P$ . Then  $p_i = r_i L$ . Therefore, if  $r_i = r_j$ , then  $p_i = p_j$ . Thus,  $r(P) \leq r(N)$ . ◀

Let  $M$  be a  $n \times m$  matrix. For  $x \in X \subseteq [n]$ ,  $y \in Y \subseteq [m]$ , we denote by  $M[X, Y]$  the  $|X| \times |Y|$  sub-matrix of  $M$ ,  $(M[x', y'])_{x' \in X, y' \in Y}$ . Denote by  $M[X, y]$  the column vector  $(M[x', y])_{x' \in X}$  and by  $M[x, Y]$  the row vector  $(M[x, y'])_{y' \in Y}$ .

For  $x \in [n]$  (resp.  $y \in [m]$ ) we say that  $M[X, y]$  is a *new column* (resp.  $M[x, Y]$  is a *new row*) to  $M[X, Y]$  if it is not equal to any of the columns (resp. rows) of  $M[X, Y]$ .

► **Lemma 5.** *Let  $M$  be a  $n \times m$  matrix,  $x \in [n]$ ,  $X \subseteq [n]$ ,  $y \in [m]$ , and  $Y \subseteq [m]$ . Suppose  $M[x, Y]$  is not a new row to  $M[X, Y]$ , and  $M[X, y]$  is not a new column to  $M[X, Y]$ . Then  $M[x, Y \cup \{y\}]$  is not a new row to  $M[X, Y \cup \{y\}]$  if and only if  $M[X \cup \{x\}, y]$  is not a new column to  $M[X \cup \{x\}, Y]$ .*

<sup>4</sup> Here  $x'$  is the transpose of  $x$ .

**Proof.** If  $M[x, Y \cup \{y\}]$  is not a new row to  $M[X, Y \cup \{y\}]$ , then there is  $x' \in X$  such that  $M[x, Y \cup \{y\}] = M[x', Y \cup \{y\}]$ . Since  $M[X, y]$  is not a new column to  $M[X, Y]$ , there is  $y' \in Y$  such that  $M[X, y] = M[X, y']$ . Since  $M[x, Y \cup \{y\}] = M[x', Y \cup \{y\}]$ , we have  $M[x', y'] = M[x, y']$  and  $M[x, y] = M[x', y]$ . Since  $M[X, y] = M[X, y']$ , we have  $M[x', y] = M[x', y']$ . Therefore,  $M[x, y] = M[x, y']$  and  $M[X \cup \{x\}, y] = M[X \cup \{x\}, y']$ . Thus,  $M[X \cup \{x\}, y]$  is not a new column to  $M[X \cup \{x\}, Y]$ .

Similarly, the other direction follows.  $\blacktriangleleft$

### 3 Matrices of $s$ -Binary Rank $d$

In this section, we prove the following two Lemmas.

► **Lemma 6.** For any  $n \times m$   $(0, 1)$ -matrix  $M$  of  $s$ -binary rank at most  $d$ , we have

$$r(M) \cdot c(M) \leq \binom{d}{\leq s} 2^d.$$

► **Lemma 7.** There is a  $(0, 1)$ -matrix  $M'$  of  $s$ -binary rank  $d$  that satisfies  $r(M') \cdot c(M') = \binom{d}{\leq s} 2^d$ .

To prove Lemma 6, we use the following Sgall's lemma.

► **Lemma 8.** [10]. Let  $\mathcal{A}, \mathcal{B} \subseteq 2^{[d]}$  be such that for every  $A \in \mathcal{A}$  and  $B \in \mathcal{B}$ ,  $|A \cap B| \leq s$ . Then  $|\mathcal{A}| \cdot |\mathcal{B}| \leq \binom{d}{\leq s} 2^d$ .

We now prove Lemma 6.

**Proof.** Since the  $s$ -binary rank of  $M$  is at most  $d$ , by Lemma 3, there is a  $n \times d$   $(0, 1)$ -matrix  $N$  and a  $d \times m$   $(0, 1)$ -matrix  $L$  such that, for  $P = NL$

1. For every  $(i, j) \in [n] \times [m]$ ,  $M[i, j] = 0$  if and only if  $P[i, j] = 0$ .
2. For every  $(i, j) \in [n] \times [m]$ ,  $P[i, j] \leq s$ .

Obviously,  $r(M) \leq r(P)$  and  $c(M) \leq c(P)$ . Consider  $\mathcal{A} = \{A_1, \dots, A_n\} \subseteq 2^{[d]}$  and  $\mathcal{B} = \{B_1, \dots, B_m\} \subseteq 2^{[d]}$ , where  $A_i = \{j | N_{i,j} = 1\}$  and  $B_k = \{j | L_{j,k} = 1\}$ . Since the entries of  $P = NL$  are at most  $s$ , for every  $i \in [n]$  and  $k \in [m]$ ,  $|A_i \cap B_k| \leq s$ .

By Lemma 4 and 8,

$$r(M) \cdot c(M) \leq r(P) \cdot c(P) \leq r(N) \cdot c(L) = |\mathcal{A}| \cdot |\mathcal{B}| \leq \binom{d}{\leq s} 2^d. \quad \blacktriangleleft$$

We now prove Lemma 7.

**Proof of Lemma 7.** Let  $N$  be a  $2^d \times d$   $(0, 1)$ -matrix where its rows contain all the vectors in  $\{0, 1\}^d$ . Let  $L$  be a  $d \times \binom{d}{\leq s}$  matrix where its columns contain all the vectors in  $\{0, 1\}^d$  of weight at most  $s$ . Obviously,  $P = NL$  is  $2^d \times \binom{d}{\leq s}$  with entries that are less than or equal to  $s$ . Define a  $2^d \times \binom{d}{\leq s}$   $(0, 1)$ -matrix  $M'$  where  $M'[i, j] = 0$  if and only if  $P[i, j] = 0$ . Then, by Lemma 3,  $M'$  is of  $s$ -binary rank at most  $d$ . We now show that  $r(M') \cdot c(M') = \binom{d}{\leq s} 2^d$ .

Since the identity  $d \times d$  matrix  $I_d$  is a sub-matrix of  $L$ , we have that  $NI_d = N$  is  $(0, 1)$ -matrix and a sub-matrix of  $P$  and therefore of  $M'$ . Therefore,  $r(M') \geq r(N) = 2^d$ . Since  $I_d$  is a sub-matrix of  $N$ , by the same argument,  $c(M') \geq c(L) = \binom{d}{\leq s}$ . Therefore  $r(M') \cdot c(M') \geq \binom{d}{\leq s} 2^d$ . Thus,  $r(M') \cdot c(M') = \binom{d}{\leq s} 2^d$ .

We now show that  $M'$  has  $s$ -binary rank  $d$ . Suppose the contrary, i.e.,  $M'$  has binary rank  $d' < d$ . Then there are  $2^d \times d'$   $(0, 1)$ -matrix  $N$  and  $d' \times \binom{d}{\leq s}$   $(0, 1)$ -matrix  $L$  such that  $P = NL$  and  $M'[i, j] = 0$  iff  $P[i, j] = 0$ . Now by Lemma 4,  $r(M') \leq r(P) \leq r(N) \leq 2^{d'} < 2^d$ , which gives a contradiction.  $\blacktriangleleft$

#### 4 Testing The $s$ -Binary Rank

In this section, we present the adaptive and non-adaptive testing algorithms for  $s$ -binary rank at most  $d$ . We first give the adaptive algorithm and prove Theorem 1.

##### 4.1 The Adaptive Tester

```

Adaptive-Test-Rank( $d, s, M, n, m, \epsilon$ )
Input: Oracle that accesses the entries of  $n \times m$  (0,1)-matrix  $M$ .
Output: Either “Accept” or “Reject”

1.  $X \leftarrow \{1\}; Y \leftarrow \{1\}; t = 9d/\epsilon$ .
2. While  $|X| \cdot |Y| \leq \binom{d}{\leq s} 2^d$  do
3.   If the  $s$ -binary rank of  $M[X, Y]$  is greater than  $d$ , then Reject.
4.    $Finish \leftarrow False; X' \leftarrow \emptyset; Y' \leftarrow \emptyset$ . /*  $X'$  and  $Y'$  are multi-sets.
5.   If  $|X| \geq |Y|$  then
6.     While (NOT  $Finish$ ) AND  $|X'| < t$ 
7.       Draw uniformly at random  $x \in [n] \setminus X; X' \leftarrow X' \cup \{x\}$ ;
8.       If  $M[x, Y]$  is a new row to  $M[X, Y]$ 
9.         then  $X \leftarrow X \cup \{x\}; Finish \leftarrow True$ .
10.    If (NOT  $Finish$ ) then
11.      While (NOT  $Finish$ ) AND  $|Y'| < t$ 
12.        Draw uniformly at random  $y \in [m] \setminus Y; Y' \leftarrow Y' \cup \{y\}$ .
13.        If  $M[X, y]$  is new column to  $M[X, Y]$ 
14.          then  $Y \leftarrow Y \cup \{y\}; Finish \leftarrow True$ .
15.    Else ( $|X| < |Y|$ )
16.      While (NOT  $Finish$ ) AND  $|Y'| < t$ 
17.        Draw uniformly at random  $y \in [m] \setminus Y; Y' \leftarrow Y' \cup \{y\}$ ;
18.        If  $M[X, y]$  is a new column to  $M[X, Y]$ 
19.          then  $Y \leftarrow Y \cup \{y\}; Finish \leftarrow True$ .
20.      If (NOT  $Finish$ ) then
21.        While (NOT  $Finish$ ) AND  $|X'| < t$ 
22.          Draw uniformly at random  $x \in [n] \setminus X; X' \leftarrow X' \cup \{x\}$ 
23.          If  $M[x, Y]$  is a new row to  $M[X, Y]$ 
24.            then  $X \leftarrow X \cup \{x\}; Finish \leftarrow True$ .
25.          While (NOT  $Finish$ ) AND  $X' \neq \emptyset$  do
26.            Draw uniformly at random  $x \in X'$  and  $y \in Y'$ 
27.            If  $M[x, Y \cup \{y\}]$  is a new row to  $M[X, Y \cup \{y\}]$  OR, equivalently,
28.               $M[X \cup \{x\}, y]$  is a new column to  $M[X \cup \{x\}, Y]$ 
29.                then  $X \leftarrow X \cup \{x\}; Y \leftarrow Y \cup \{y\}; Finish \leftarrow True$ .
30.                else  $X' \leftarrow X' \setminus \{x\}; Y' \leftarrow Y' \setminus \{y\}$ .
31.            If (NOT  $Finish$ ) then Accept
32.            else Reject

```

■ **Figure 1** An adaptive tester for  $s$ -binary rank at most  $d$ .

In this section, we prove Theorem 1.

Consider the tester **Adaptive-Test-Rank** in Figure 1. The tester, at every iteration of the main While-loop (step 2) has a set  $X$  of rows of  $M$  and a set  $Y$  of columns of  $M$ . If  $|X| \geq |Y|$  (step 5), the tester first tries to extend  $M[X, Y]$  with a new column (steps 6-8). If it succeeds, it moves to the next iteration. Otherwise, it tries to extend  $M[X, Y]$  with a new row (steps 9-12). If it succeeds, it moves to the next iteration. Otherwise, it tries to extend  $M[X, Y]$  with a new row and a new column (steps 21-26). If it succeeds, it moves to the next iteration. If it fails, it accepts (step 27). If  $|X| < |Y|$  (step 13), it starts with the row of  $M[X, Y]$  (steps 14-16), then the column (steps 18-20), and then both (steps 21-26). If it fails, it accepts (step 27).

If  $|X| \cdot |Y| > \binom{d}{\leq s} 2^d$  (step 2 and then step 28) or the  $s$ -binary rank of  $M[X, Y]$  is greater than  $d$  (step 3), then it rejects.

We first prove

► **Lemma 9.** *Let  $t = 9d/\epsilon$ . Tester **Adaptive-Test-Rank** makes at most  $2 \binom{d}{\leq s} 2^{dt} = \tilde{O} \left( \binom{d}{\leq s} 2^d \right) / \epsilon$  queries.*

**Proof.** We prove by induction that at every iteration of the main While-loop (step 2), the tester knows the entries of  $M[X, Y]$ , and the total number of queries,  $q_{X,Y}$ , is at most  $2|X||Y|t$ . Since the While-loop condition is  $|X||Y| \leq \binom{d}{\leq s} 2^d$ , the result follows.

At the beginning of the algorithm, no queries are made, and  $|X| = |Y| = 1$ . Then  $2|X||Y|t = 2t > 0 = q_{X,Y}$ . Suppose, at the  $k$ th iteration, the tester knows the entries of  $M[X, Y]$  and  $q_{X,Y} \leq 2|X||Y|t$ . We prove the result for the  $(k+1)$ th iteration.

We have the following cases (at the  $(k+1)$ th iteration)

**Case I.**  $|X| \geq |Y|$  (step 5) and, for some  $x$ ,  $M[x, Y]$  is a new row to  $M[X, Y]$  (step 8).

In that case, *Finish* becomes *true*, and no other sub-while-loop is executed. Therefore, the number of queries made at this iteration is at most  $|Y|t$  (to find all  $M[x, Y]$ ), and one element  $x$  is added to  $X$ . Then, the tester knows all the entries of  $M[X \cup \{x\}, Y]$  and

$$q_{X \cup \{x\}, Y} = q_{X, Y} + |Y|t \leq 2|X||Y|t + |Y|t \leq 2|X \cup \{x\}| \cdot |Y|t,$$

and the result follows.

**Case II.**  $|X| \geq |Y|$  (step 5), for all  $x' \in X'$ ,  $M[x', Y]$  is not a new row to  $M[X, Y]$  (step 8), and for some  $y$ ,  $M[X, y]$  is a new column to  $M[X, Y]$  (step 12).

In that case, *Finish* becomes *true*, and no other sub-while-loop is executed after the second sub-while-loop (step 10).

Therefore, in this case, the number of queries made at this iteration is at most  $|Y|t + |X|t$ .  $|X|t$  queries in the first sub-while-loop (to find  $M[x, Y]$  for all  $x \in X'$ ), and at most  $|Y|t$  queries in the second sub-while-loop (to find  $M[X, y']$  for all  $y' \in Y'$ ). Then one element  $y$  is added to  $Y$ . Therefore, the tester knows the entries of  $M[X, Y \cup \{y\}]$  and, since  $|Y| \leq |X|$ ,

$$q_{X, Y \cup \{y\}} = q_{X, Y} + |X|t + |Y|t \leq 2|X||Y|t + 2|X|t = 2|X| \cdot |Y \cup \{y\}|t,$$

and the result follows.

**Case III.**  $|X| \geq |Y|$ , for all  $x' \in X'$ ,  $M[x', Y]$  is not a new row to  $M[X, Y]$ , for all  $y' \in Y'$ ,  $M[X, y']$  is not a new column to  $M[X, Y]$ , and for some  $x \in X'$ ,  $y \in Y'$ ,  $M[x, Y \cup \{y\}]$  is a new row to  $M[X, Y \cup \{y\}]$  (step 23).

## 27:8 On Property Testing of the Binary Rank

In this case,  $|X'| = |Y'| = t$ , the number of queries is  $|X|t + |Y|t + t$ . Exactly  $|X|t$  queries in the first sub-while-loop,  $|Y|t$  queries in the second sub-while-loop, and at most<sup>5</sup>  $t$  queries in the sub-while-loop in step 21. Then one element  $x$  is added to  $X$ , and one element  $y$  is added to  $Y$ . Then the tester knows the entries of  $M[X \cup \{x\}, Y \cup \{y\}]$  and

$$q_{X \cup \{x\}, Y \cup \{y\}} = q_{X, Y} + |X|t + |Y|t + t \leq 2|X| \cdot |Y|t + |X|t + |Y|t + t \leq 2|X \cup \{x\}| \cdot |Y \cup \{y\}|t.$$

**Case IV.**  $|X| \geq |Y|$ , for all  $x' \in X'$ ,  $M[x', Y]$  is not a new row to  $M[X, Y]$ , for all  $y' \in Y'$ ,  $M[X, y']$  is not a new column to  $M[X, Y]$ , and for all the drawn pairs  $x \in X', y \in Y'$ ,  $M[x, Y \cup \{y\}]$  is not a new row to  $M[X, Y \cup \{y\}]$  (step 23).

In this case, *Finish* will have value *False*, and the tester accepts in step 27.

The analysis of the case when  $|X| < |Y|$  is similar to the above analysis. ◀

We now prove the completeness of the tester.

► **Lemma 10.** *If  $M$  is a  $n \times m$   $(0, 1)$ -matrix of  $s$ -binary rank at most  $d$ , then the tester **Adaptive-Test-Rank** accepts with probability 1.*

**Proof.** The tester rejects if and only if one of the following occurs,

1.  $M[X, Y]$  has  $s$ -binary rank greater than  $d$ .
2.  $|X| \cdot |Y| > \binom{d}{\leq s} 2^d$ .

If  $M[X, Y]$  has  $s$ -binary rank greater than  $d$ , then  $M$  has  $s$ -binary rank greater than  $d$ . This is because, if  $M = NL$ , then  $M[X, Y] = N[X, [d]] \cdot L[[d], Y]$ . So item 1 cannot occur.

Before we show that item 2 cannot occur, we prove the following:

▷ **Claim 11.** The rows (resp. columns) of  $M[X, Y]$  are distinct.

**Proof.** The steps in the tester where we add rows or columns are steps 8, 12 16, 20, and 23. In steps 8, 12 16, 20 it is clear that a row (resp. column) is added only if it is a new row (resp. column) to  $M[X, Y]$ . Consider step 23 and suppose, w.l.o.g  $|X| \geq |Y|$ . This step is executed only when *Finish* = *False*. This happens when  $|X'| = |Y'| = t$ , for every  $x \in X'$ ,  $M[x, Y]$  is not a new row to  $M[X, Y]$ , and for every  $y \in Y'$ ,  $M[X, y]$  is not a new column to  $M[X, Y]$ . Then  $x$  and  $y$  are added to  $X$  and  $Y$ , respectively, if  $M[x, Y \cup \{y\}]$  is a new row to  $M[X, Y \cup \{y\}]$ . Then, by Lemma 5,  $M[X \cup \{x\}, y]$  is a new column to  $M[X \cup \{x\}, Y]$ . So, the rows (and columns) in  $M[X \cup \{x\}, Y \cup \{y\}]$  are distinct. This implies the result. ◀

Suppose, to the contrary,  $|X| \cdot |Y| > \binom{d}{\leq s} 2^d$ . Since  $M' = M[X, Y]$  satisfies  $r(M')c(M') = |X| \cdot |Y| > \binom{d}{\leq s} 2^d$ , by Lemma 6, the  $s$ -binary rank of  $M'$ , and therefore of  $M$ , is greater than  $d$ . A contradiction. ◀

We now prove the soundness of the tester.

We first prove the following.

▷ **Claim 12.** Let  $M$  be a  $n \times m$   $(0, 1)$ -matrix,  $X \subseteq [n]$ , and  $Y \subseteq [m]$ . Suppose there are two functions,  $' : [n] \rightarrow X$  and  $'' : [m] \rightarrow Y$ , such that

1. For every  $x \in [n]$ ,  $M[x, Y] = M[x', Y]$ .
2. For every  $y \in [m]$ ,  $M[X, y] = M[X, y'']$ .
3. For every  $x \in [n]$  and  $y \in [m]$ ,  $M[x, y] = M[x', y'']$ .

Then  $M$  has at most  $|X|$  distinct rows and  $|Y|$  distinct columns, and its  $s$ -binary rank is the  $s$ -binary rank of  $M[X, Y]$ .

<sup>5</sup> This is because, for  $x \in X', y \in Y'$ , the tester already knows  $M[x, Y]$  and  $M[X, y]$  from the first and second sub-while-loop and only needs to query  $M[x, y]$ .

Proof. Let  $x \in [n] \setminus X$ . For every  $y$ ,  $M[x, y] = M[x', y''] = M[x', y]$ . Therefore, row  $x$  in  $M$  is equal to row  $x'$ . Similarly, column  $y$  in  $M$  is equal to column  $y''$ .

Since adding equal columns and rows to a matrix does not change the  $s$ -binary rank<sup>6</sup>, we have  $\text{br}_s(M[X, Y]) = \text{br}_s(M[X, [m]]) = \text{br}_s(M)$ .  $\triangleleft$

The following Claim is proved in [9] (Claim 10). Here, we give the proof for completeness.

$\triangleright$  **Claim 13.** Let  $M$  be a  $(0, 1)$ -matrix that is  $\epsilon$ -far from having  $s$ -binary rank at most  $d$ . Let  $X \subseteq [n]$  and  $Y \subseteq [m]$ , such that  $\text{br}_s(M[X, Y]) \leq d$ , the columns of  $M[X, Y]$  are distinct, and the rows of  $M[X, Y]$  are distinct. Then one of the following must hold:

1. The number of rows  $x \in [n]$  where  $M[x, Y]$  is a new row to  $M[X, Y]$  is at least  $n\epsilon/3$ .
2. The number of columns  $y \in [m]$  where  $M[X, y]$  is a new column to  $M[X, Y]$  is at least  $m\epsilon/3$ .
3. The number pairs  $(x, y)$ ,  $x \notin X$ ,  $y \notin Y$ , where,  $M[x, Y] = M[x', Y]$  for some  $x' \in X$ ,  $M[X, y] = M[X, y'']$  for some  $y'' \in Y$ , and  $M[x, y] \neq M[x', y'']$ , is at least  $mn\epsilon/3$ .

Proof. Assume, to the contrary, that none of the above statements holds. Change every row  $x$  in  $M$  where  $M[x, Y]$  is a new row to  $M[X, Y]$  to a zero row. Let  $X'$  be the set of such rows. Change every column  $y$  in  $M$  where  $M[X, y]$  is a new column to  $M[X, Y]$  to a zero column. Let  $Y'$  be the set of such columns. For every other entry  $(x, y)$ ,  $x \notin X$ ,  $y \notin Y$  that is not changed to zero and  $M[x, y] \neq M[x', y'']$ , change  $M[x, y]$  to  $M[x', y'']$ . Let  $M'$  be the matrix obtained from the above changes.

The number of entries  $(x, y)$  where  $M[x, y] \neq M'[x, y]$  is less than  $(n\epsilon/3)m + (m\epsilon/3)n + mn\epsilon/3 = \epsilon mn$ . Therefore,  $M'$  is  $\epsilon$ -close to  $M$ . By claim 13,  $\text{br}_s(M') = \text{br}_s(M[[n] \setminus X', [m] \setminus Y']) = \text{br}_s(M[X, Y]) \leq d$ . A contradiction.  $\triangleleft$

We now prove the soundness of the tester.

$\blacktriangleright$  **Lemma 14.** *If  $M$  is  $\epsilon$ -far from having  $s$ -binary rank  $d$ , then with probability at least  $2/3$ , Adaptive-Test-Rank rejects.*

**Proof.** Consider the while-loop in step 2 at some iteration  $i$ . If  $\text{br}_s(M[X, Y]) > d$ , then the tester rejects in step 3. We will now show that if  $\text{br}_s(M[X, Y]) \leq d$ , then, with probability at most  $3e^{-2d}$ , the tester accepts at iteration  $i$ .

To this end, let  $\text{br}_s(M[X, Y]) \leq d$ . Then, by Claim 13, one of the following holds.

1. The number of rows  $x \in [n]$  where  $M[x, Y]$  is a new row to  $M[X, Y]$  is at least  $n\epsilon/3$ .
2. The number of columns  $y \in [m]$  where  $M[X, y]$  is a new column to  $M[X, Y]$  is at least  $m\epsilon/3$ .
3. The number pairs  $(x, y)$ ,  $x \notin X$ ,  $y \notin Y$ , where,  $M[x, Y] = M[x', Y]$  for some  $x' \in X$ ,  $M[X, y] = M[X, y'']$  for some  $y'' \in Y$ , and  $M[x, y] \neq M[x', y'']$ , is at least  $mn\epsilon/3$ .

Now at the  $i$ th iteration, suppose w.l.o.g,  $|X| \geq |Y|$  (the other case  $|Y| < |X|$  is similar). If item 1 occurs, then with probability at least  $p = 1 - (1 - \epsilon/3)^t \geq 1 - e^{-2d}$ , the tester finds a new row to  $M[X, Y]$  and does not accept at iteration  $i$ . If item 2 occurs, then if it does not find a new row to  $M[X, Y]$ , with probability at least  $p$ , the tester finds a new column to  $M[X, Y]$  and does not accept. If item 3 occurs, and it does not find a new row or column to  $M[X, Y]$ , then with probability at least  $p$ , it finds such a pair and does not accept. Therefore, with probability at most  $3(1 - p) \leq 3e^{-2d}$ , the tester accepts at iteration  $i$ .

<sup>6</sup> If we add a column to a matrix that is equal to column  $y$ , then the rectangles that cover column  $y$  can be extended to cover the added column.

## 27:10 On Property Testing of the Binary Rank

Since the while-loop runs at most  $|X| + |Y| \leq 2|X||Y| \leq 2\binom{d}{\leq s}2^d \leq 2^{2d+1}$  iterations, with probability at most  $3e^{-2^d}2^{2d+1} \leq 1/3$ , the tester accepts in while-loop. Therefore, with probability at least  $2/3$ , the tester does not accept in the while-loop. Thus, it either rejects because  $\text{br}_s(M[X, Y]) > d$  or rejects in step 28. ◀

### 4.2 The Non-Adaptive Tester

In this section, we prove Theorem 2.

**Non-Adaptive-Test-Rank**( $d, s, M, n, m, \epsilon$ )

**Input:** Oracle that accesses the entries of  $(0, 1)$ -matrix  $M$ .

**Output:** Either “Accept” or “Reject”.

1.  $T \leftarrow \frac{324 \cdot d^2 \binom{d}{\leq s} 2^d}{\epsilon^2}$ .
2. Draw uniformly at random  $x^{(1)}, \dots, x^{(T)} \in [n]$ .
3. Draw uniformly at random  $y^{(1)}, \dots, y^{(T)} \in [m]$ .
4. For every  $i \in [T]$  and  $j \in [T]$  such that  $i \cdot j \leq T$
5.  $D[i, j] \leftarrow \text{Query } M[x^{(i)}, y^{(j)}]$
6.  $u = 1; w = 1$ .
7. Run **Adaptive-Test-Rank**( $d, s, M, n, m, \epsilon$ )
  - When the tester asks for a uniform at random  $x$  - return  $x^{(u)}$ ;  $u \leftarrow u + 1$
  - When the tester asks for a uniform at random  $y$  - return  $y^{(w)}$ ;  $w \leftarrow w + 1$
  - When the tester makes the Query  $M[x^{(i)}, y^{(j)}]$  - return  $D[i, j]$

■ **Figure 2** A non-adaptive tester for  $s$ -binary rank at most  $d$ .

First, consider **Adaptive-Test-Rank** in Figure 1. Consider steps 7, 11, 15, and 19, where it draws a new column or row. We prove.

► **Lemma 15.** *Let  $t = 9d/\epsilon$ . At each iteration of **Adaptive-Test-Rank**, the total number of uniformly at random rows  $x \in [n]$  drawn is at most  $(|X| + \min(|X|, |Y| - 1))t$ , and the number of uniformly at random rows  $y \in [m]$  drawn is at most  $(|Y| + \min(|X|, |Y|))t$ .*

**Proof.** We prove by induction that at every iteration of the main While-loop (step 2), the total number of random rows drawn by the tester,  $n_{X, Y}$ , is at most  $(|X| + \min(|X|, |Y| - 1))t$ , and the total number of random columns drawn,  $m_{X, Y}$ , is at most  $(|Y| + \min(|X|, |Y|))t$ .

At the beginning,  $|X| = |Y| = 1$ , and the number of columns and rows is 1. In that case,<sup>7</sup>  $n_{X, Y} = 1 \leq t$  and  $m_{X, Y} = 1 \leq 2t$ . Suppose, at the  $k$ th iteration, the induction statement is true. We prove the result for the  $(k + 1)$ th iteration.

At the  $(k + 1)$ th iteration, we have the following cases.

**Case I.**  $|X| \geq |Y|$  (step 5) and, for some  $x$ ,  $M[x, Y]$  is a new row to  $M[X, Y]$  (step 8).

In that case, *Finish* becomes *true*, and no other sub-while-loop is executed. Therefore, the number of rows drawn at this iteration is at most  $t$ , and one element  $x$  is added to  $X$ . No columns are drawn. Then,

$$n_{X \cup \{x\}, Y} \leq n_{X, Y} + t \leq (|X| + \min(|X|, |Y| - 1) + 1)t \leq (|X \cup \{x\}| + \min(|X \cup \{x\}|, |Y| - 1))t,$$

<sup>7</sup> We assume that the first column/row drawn is column/row one

and

$$m_{X \cup \{x\}, Y} = m_{X, Y} \leq (|Y| + \min(|X|, |Y|))t \leq (|Y| + \min(|X \cup \{x\}|, |Y|))t.$$

Thus, the result follows for this case.

**Case II.**  $|X| \geq |Y|$  (step 5), for all  $x' \in X'$ ,  $M[x', Y]$  is not a new row to  $M[X, Y]$  (step 8), and for some  $y$ ,  $M[X, y]$  is a new column to  $M[X, Y]$  (step 12).

In that case, *Finish* becomes *true*, and no other sub-while-loop is executed after the second sub-while-loop (step 10).

Therefore, in this case, the number of rows drawn at this iteration is  $t$ , one element  $y$  is added to  $Y$ , and the number of columns drawn is at most  $t$ . Then

$$\begin{aligned} n_{X, Y \cup \{y\}} = n_{X, Y} + t &\leq (|X| + \min(|X|, |Y| - 1) + 1)t \\ &= (|X| + |Y|)t = (|X| + \min(|X|, |Y \cup \{y\}| - 1))t, \end{aligned}$$

and

$$m_{X, Y \cup \{y\}} \leq m_{X, Y} + t \leq (|Y| + \min(|X|, |Y|) + 1)t \leq (|Y \cup \{y\}| + \min(|X|, |Y \cup \{y\}|))t.$$

Thus, the result follows for this case.

**Case III.**  $|X| < |Y|$  (step 13), and for some  $y$ ,  $M[X, y]$  is a new column to  $M[X, Y]$  (step 16).

In that case, *Finish* becomes *true*, and no other sub-while-loop is executed. Therefore, the number of columns drawn at this iteration is at most  $t$ , and one element  $y$  is added to  $Y$ . No rows are drawn. Then,

$$n_{X, Y \cup \{y\}} = n_{X, Y} \leq (|X| + \min(|X|, |Y| - 1))t \leq (|X| + \min(|X|, |Y \cup \{y\}| - 1))t,$$

and

$$m_{X, Y \cup \{y\}} \leq m_{X, Y} + t \leq (|Y| + \min(|X|, |Y|) + 1)t = (|Y \cup \{y\}| + \min(|X|, |Y \cup \{y\}|))t.$$

Thus, the result follows for this case.

**Case IV.**  $|X| < |Y|$  (step 13), for all  $y' \in Y'$ ,  $M[X, y']$  is not a new row to  $M[X, Y]$ , and for some  $x$ ,  $M[x, Y]$  is a new column to  $M[X, Y]$  (step 20). In that case, *Finish* becomes *true*, and no other sub-while-loop is executed after the fourth sub-while-loop (step 18).

In this case, the number of rows drawn at this iteration is  $t$ , one element  $x$  is added to  $X$ , and the number of columns drawn is at most  $t$ . Then

$$\begin{aligned} n_{X \cup \{x\}, Y} = n_{X, Y} + t &\leq (|X| + \min(|X|, |Y| - 1) + 1)t \\ &\leq (|X \cup \{x\}| + \min(|X \cup \{x\}|, |Y| - 1))t \\ m_{X \cup \{x\}, Y} &\leq m_{X, Y} + t \leq (|Y| + \min(|X|, |Y|) + 1)t = (|Y| + \min(|X \cup \{x\}|, |Y|))t. \end{aligned}$$

Thus, the result follows for this case.

**Case V.** For all  $x' \in X'$ ,  $M[x', Y]$  is not a new row to  $M[X, Y]$ , for all  $y' \in Y'$ ,  $M[X, y']$  is not a new column to  $M[X, Y]$ , and for some  $x \in X'$ ,  $y \in Y'$ ,  $M[x, Y \cup \{y\}]$  is a new row to  $M[X, Y \cup \{y\}]$  (step 23).

In this case, the number of rows drawn at this iteration is  $t$ , the number of columns drawn is  $t$ , one element  $x$  is added to  $X$ , and one element  $y$  is added to  $Y$ . Then

$$\begin{aligned} n_{X \cup \{x\}, Y \cup \{y\}} = n_{X, Y} + t &\leq (|X| + \min(|X|, |Y| - 1) + 1)t \\ &\leq (|X \cup \{x\}| + \min(|X \cup \{x\}|, |Y \cup \{y\}| - 1))t. \end{aligned}$$

$$\begin{aligned} m_{X \cup \{x\}, Y \cup \{y\}} = m_{X, Y} + t &\leq (|Y| + \min(|X|, |Y|) + 1)t \\ &\leq (|Y \cup \{y\}| + \min(|X \cup \{x\}|, |Y \cup \{y\}|))t. \end{aligned} \quad \blacktriangleleft$$

## 27:12 On Property Testing of the Binary Rank

We are now ready to prove Theorem 2.

**Proof.** By Lemma 15, the total number of rows and columns drawn in **Adaptive-Test-Rank** up to iteration  $t$  is at most  $n' := 9(|X| + \min(|X|, |Y| - 1))d/\epsilon \leq 18|X|d/\epsilon$  and  $m' := 9(|Y| + \min(|X|, |Y|))d/\epsilon \leq 18|Y|d/\epsilon$ , respectively. We also have  $|X| \cdot |Y| \leq \binom{d}{\leq s} 2^d$ . So

$$n' \cdot m' \leq 324|X||Y|d^2/\epsilon^2 \leq T := \frac{324 \cdot d^2 \binom{d}{\leq s} 2^d}{\epsilon^2}.$$

Consider the tester **Non-Adaptive-Test-Rank** in Figure 2. The tester draws  $T$  rows  $x^{(1)}, \dots, x^{(T)} \in [n]$ , and columns  $y^{(1)}, \dots, y^{(T)} \in [m]$  and queries all  $M[x^{(i)}, y^{(j)}]$  where  $ij \leq T$  and puts the result in the table  $D$ . Then it runs **Adaptive-Test-Random** using the above-drawn rows and columns. We now show that all the queries that **Adaptive-Test-Random** makes can be fetched from the table  $D$ .

At any iteration, the number of rows drawn is at most  $n'$ , and the number of rows drawn is at most  $m'$ . Therefore, the tester needs to know (in the worst case) all the entries  $M[x^{(i)}, y^{(j)}]$  where  $i \leq n'$  and  $j \leq m'$ . Since  $ij \leq n'm' \leq T$ , the result follows.

The number of queries that the tester makes is

$$\sum_{i=1}^T \frac{T}{i} = O(T \ln T) = \tilde{O}\left(\frac{\binom{d}{\leq s} 2^d}{\epsilon^2}\right). \quad \blacktriangleleft$$

### 5 Testing the Exact $s$ -Binary Rank

We first prove the following.

► **Lemma 16.** *Let  $M$  and  $M'$  be  $n \times m$   $(0, 1)$ -matrices that differ in one row (or column). Then  $|\text{br}_s(M) - \text{br}_s(M')| \leq 1$ .*

**Proof.** Suppose  $\text{br}_s(M) = d$  and  $M'$  differ from  $M$  in row  $k$ . Let  $N$  and  $L$  be  $n \times d$   $(0, 1)$ -matrix and  $d \times m$   $(0, 1)$ -matrix, respectively, such that  $P = NL$ , for every  $(i, j) \in [n] \times [m]$ ,  $P[i, j] \leq s$ , and  $P[i, j] = 0$  if and only if  $M[i, j] = 0$ . Add to  $N$  a column (as a  $(d + 1)$ th column) that all its entries are zero except the  $k$ -th entry, which equals 1. Then change  $N[k, j]$  to zero for all  $j \in [d]$ . Let  $N'$  be the resulting matrix. Add to  $L$  another row (as a  $(d + 1)$ th row) equal to the  $k$ -th row of  $M'$ . Let  $L'$  be the resulting matrix. Let  $P' = N'L'$ . It is easy to see that  $P'[i, j] = P[i, j]$  for all  $i \neq k$  and  $j$ , and the  $k$ th row of  $P'$  is equal to the  $k$ th row of  $M'$ . Then, for every  $(i, j) \in [n] \times [m]$ ,  $P'[i, j] \leq s$ , and  $P'[i, j] = 0$  if and only if  $M'[i, j] = 0$ . Therefore,  $\text{br}_s(M') \leq d + 1 = \text{br}_s(M) + 1$ . In the same way,  $\text{br}_s(M) \leq \text{br}_s(M') + 1$ .  $\blacktriangleleft$

► **Lemma 17.** *Let  $\eta = d^2/(nm)$ . Let  $M$  be  $n \times m$   $(0, 1)$ -matrix. If  $M$  is  $\epsilon$ -close to having  $s$ -binary rank at most  $d$ , then  $M$  is  $(\epsilon + \eta)$ -close to having  $s$ -binary rank  $d$ .*

**Proof.** We will show that for every  $n \times m$   $(0, 1)$ -matrix  $H$  of  $s$ -binary rank at most  $d - 1$ , there is a  $n \times m$   $(0, 1)$ -matrix  $G$  of  $s$ -binary rank  $d$  that is  $\eta$ -close to  $H$ . Therefore, if  $M$  is  $\epsilon$ -close to having  $s$ -binary rank at most  $d$ , then it is  $(\epsilon + \eta)$ -close to having  $s$ -binary rank  $d$ .

Define the  $n \times m$   $(0, 1)$ -matrices  $G_k$ ,  $k \in [d] \cup \{0\}$ , where  $G_0 = H$  and for  $k \geq 1$ ,  $G_k[i, j] = H[i, j]$  if  $j > k$  or  $i > d$ , and  $G_k[[d], [k]] = I_d[[d], [k]]$  where  $I_d$  is the  $d \times d$  identity matrix. Since  $G_d[[d], [d]] = I_d$ , we have  $\text{br}_s(G_d) \geq d$ . It is clear that for every  $k \in [d] \cup \{0\}$ ,  $G_k$  is  $(d^2/nm)$ -close to  $H$ . If  $\text{br}_s(G_d) = d$ , then take  $G = G_d$ , and we are done. Otherwise, suppose  $\text{br}_s(G_d) > d$ .

Now consider a sequence  $H = G_0, G_1, G_2, \dots, G_d$ . By Lemma 16, we have  $\text{br}_s(G_{i-1}) - 1 \leq \text{br}_s(G_i) \leq \text{br}_s(G_{i-1}) + 1$ . Now since  $\text{br}_s(G_0) = \text{br}_s(H) \leq d - 1$  and  $\text{br}_s(G_d) > d$ , by the discrete intermediate value theorem, there must be  $k \in [d]$  such that  $\text{br}_s(G_k) = d$ . Then take  $G = G_k$ , and we are done.  $\blacktriangleleft$

Now, the tester for testing the  $s$ -binary rank  $d$  runs as follows. If  $mn < 2d^2/\epsilon$ , then find all the entries of  $M$  with  $mn < 2d^2/\epsilon$  queries. If  $\text{br}_s(M) = d$ , then accept. Otherwise, reject. If  $mn \geq 2d^2/\epsilon$ , then run **Adaptive-Test-Rank**( $d, s, M, n, m, \epsilon/2$ ) (for the non-adaptive, we run **Non-Adaptive-Test-Rank**( $d, s, M, n, m, \epsilon/2$ )) and output its answer.

We now show the correctness of this algorithm. If  $M$  is of  $s$ -binary rank  $d$ , then it is of  $s$ -binary rank at most  $d$ , and the tester accepts.

Now, suppose  $f$  is  $\epsilon$ -far from having  $s$ -binary rank  $d$ . If  $mn < 2d^2/\epsilon$ , the tester rejects. If  $mn \geq 2d^2/\epsilon$ , then, by Lemma 17,  $f$  is  $(\epsilon - \eta)$ -far from having  $s$ -binary rank at most  $d$ , where  $\eta = d^2/(nm)$ . Since  $\eta = d^2/(nm) \leq \epsilon/2$ , the function  $f$  is  $(\epsilon/2)$ -far from having  $s$ -binary rank at most  $d$ , and therefore the tester, with probability at least  $2/3$ , rejects.

## 6 Conclusion and Open Problems

In this work, we introduced the notion of  $s$ -binary rank for  $(0, 1)$ -matrices, extending the concept of binary rank. We established a tight upper bound on the size of matrices with  $s$ -binary rank at most  $d$ , and showed the existence of matrices achieving this bound. Using this result, we presented novel one-sided adaptive and non-adaptive testers for  $(0, 1)$ -matrices with  $s$ -binary rank at most  $d$ , significantly improving the query complexity compared to prior work. The adaptive tester requires  $\tilde{O}\left(\binom{d}{\leq s} 2^d/\epsilon\right)$  queries, while the non-adaptive tester requires  $\tilde{O}\left(\binom{d}{\leq s} 2^d/\epsilon^2\right)$  queries.

The following are open problems that are worth investigating:

**Tighter Bounds on Query Complexity:** Investigate whether the query complexity of the testers for  $(0, 1)$ -matrices with  $s$ -binary rank at most  $d$  can be further improved. Specifically, explore alternative approaches or refinements that can reduce the dependence on  $\binom{d}{\leq s}$  and  $2^d$  in the query complexity bounds.

**Generalization Beyond  $(0, 1)$ -Matrices:** Extend the concept of  $s$ -binary rank to other types of matrices, such as integer-valued matrices or matrices with entries from a larger alphabet. Study the properties, computational aspects, and property testing of these generalizations.

Addressing these open problems will lead to a more profound understanding of the  $s$ -binary rank, provide further insights into the structure of matrices, and potentially lead to improved algorithmic techniques and applications in various fields.

---

## References

- 1 Jérôme Amilhastre, Marie-Catherine Vilarem, and Philippe Janssen. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discret. Appl. Math.*, 86(2-3):125–144, 1998. doi:10.1016/S0166-218X(98)00039-0.
- 2 Maria-Florina Balcan, Yi Li, David P. Woodruff, and Hongyang Zhang. Testing matrix rank, optimally. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 727–746, 2019. doi:10.1137/1.9781611975482.46.
- 3 Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014 – 22th Annual European Symposium, Wrocław, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2014. doi:10.1007/978-3-662-44777-2\_20.

## 27:14 On Property Testing of the Binary Rank

- 4 Dana Ron. Private Communication.
- 5 David A. Gregory, Norman J. Pullman, Kathryn F. Jones, and J. Richard Lundgren. Biclique coverings of regular bigraphs and minimum semiring ranks of regular matrices. *J. Comb. Theory, Ser. B*, 51(1):73–89, 1991. doi:10.1016/0095-8956(91)90006-6.
- 6 Robert Krauthgamer and Ori Sasson. Property testing of data dimensionality. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 18–27, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644112>.
- 7 Yi Li, Zhengyu Wang, and David P. Woodruff. Improved testing of low rank matrices. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 691–700, 2014. doi:10.1145/2623330.2623736.
- 8 Yonatan Nakar and Dana Ron. On the testability of graph partition properties. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPICs*, pages 53:1–53:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.APPROX-RANDOM.2018.53.
- 9 Michal Parnas, Dana Ron, and Adi Shraibman. Property testing of the boolean and binary rank. *Theory Comput. Syst.*, 65(8):1193–1210, 2021. doi:10.1007/s00224-021-10047-8.
- 10 Jiří Sgall. Bounds on pairs of families with restricted intersections. *Comb.*, 19(4):555–566, 1999. doi:10.1007/s004939970007.

# Short Definitions in Constraint Languages

Jakub Bulín   

Department of Theoretical Computer Science and Mathematical Logic,  
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Michael Kompatscher   

Department of Algebra, Faculty of Mathematics and Physics,  
Charles University, Prague, Czech Republic

---

## Abstract

A first-order formula is called *primitive positive* (*pp*) if it only admits the use of existential quantifiers and conjunction. Pp-formulas are a central concept in (fixed-template) constraint satisfaction since  $\text{CSP}(\Gamma)$  can be viewed as the problem of deciding the primitive positive theory of  $\Gamma$ , and pp-definability captures gadget reductions between CSPs.

An important class of tractable constraint languages  $\Gamma$  is characterized by having *few subpowers*, that is, the number of  $n$ -ary relations pp-definable from  $\Gamma$  is bounded by  $2^{p(n)}$  for some polynomial  $p(n)$ . In this paper we study a restriction of this property, stating that every pp-definable relation is definable by a pp-formula of polynomial length. We conjecture that the existence of such *short definitions* is actually equivalent to  $\Gamma$  having few subpowers, and verify this conjecture for a large subclass that, in particular, includes all constraint languages on three-element domains. We furthermore discuss how our conjecture imposes an upper complexity bound of co-NP on the subpower membership problem of algebras with few subpowers.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Complexity theory and logic

**Keywords and phrases** constraint satisfaction, primitive positive definability, few subpowers, polynomially expressive, relational clone, subpower membership

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.28

**Related Version** *Previous Version:* <https://arxiv.org/abs/2305.01984v1>

**Funding** *Jakub Bulín:* Supported by the Charles University project UNCE/SCI/004 and the MŠMT ČR INTER-EXCELLENCE project LTAUSA19070.

*Michael Kompatscher:* Supported by the Charles University project UNCE/SCI/022 and the MŠMT ČR INTER-EXCELLENCE project LTAUSA19070.

**Acknowledgements** The authors would like to thank Dmitriy Zhuk for inspiring discussions about critical relations and the anonymous reviewers for their valuable suggestions.

## 1 Introduction

Constraint satisfaction is a unifying framework for expressing a wide range of computational tasks coming from a smorgasbord of real-life applications and theoretical contexts. In a CSP instance, the goal is to assign values to variables subject to a list of *constraints* to be satisfied. In the most general setting, a constraint consists of a tuple of variables (its *scope*) and a list of admissible evaluations of the scope (i.e., tuples of values, forming the *constraint relation*). Usually, the set of variables, the sets of admissible values for every variable (its *domain*), and the list of input constraints are all finite. This simple formulation strikes a “perfect balance between generality and structure” [3].

In this general formulation, the CSP is an NP-complete problem: for example, SAT or graph 3-colorability are easily expressible in this framework. However, many problems subsumed by it are tractable, e.g., 2-SAT, Horn-SAT, or checking the consistency of a system of linear equations over  $\mathbb{Z}_p$ . A natural way to explore the complex landscape of the CSP,



© Jakub Bulín and Michael Kompatscher;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 28; pp. 28:1–28:15

Leibniz International Proceedings in Informatics



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

justified by applications as well as theory [13], is to fix a finite domain  $A$  and a finite set of relations  $\Gamma$  on  $A$  that are allowed to appear as constraints (a *constraint language*) [25]; such fragment of the CSP is usually denoted by  $\text{CSP}(\Gamma)$ . (Sometimes, constraint languages on infinite domains, or with infinitely many relations are considered. But for simplicity, following, e.g., [3], we keep the two standard finiteness assumptions throughout the paper.)

The CSP dichotomy theorem [10, 27, 28] states that for every constraint language  $\Gamma$ ,  $\text{CSP}(\Gamma)$  is in P or NP-complete. To tame the vast landscape of constraint languages, it was immensely helpful to realize that various ad hoc “gadget” complexity reductions share a common explanation using the notion of *primitive positive (pp-) definability* (i.e., the usual first-order logic definability restricted to  $\{\exists, \wedge, =\}$ -formulas) [19, 18] and the more general notions of *pp-interpretability* and *pp-constructibility* [4]. In fact, the CSP dichotomy theorem implies that  $\text{CSP}(\Gamma)$  is NP-complete if and only if  $\Gamma$  pp-constructs *every* finite constraint language. Moreover, pp-definability and its generalizations have an external characterization via so-called *polymorphisms* (“multivariate homomorphisms”) [15, 7, 17]. For an introduction to the area, see [3].

Constraint languages  $\Gamma$  for which  $\text{CSP}(\Gamma)$  is solvable by a certain algorithmic approach involving computing with *compact representations* of solution sets (generalizing *bases* of vector spaces or *strong generating sets* from the Schreier-Sims algorithm for permutation groups [26]) were characterized in [6, 16] as those that have *few subpowers*, that is, the number of  $n$ -ary relations pp-definable from  $\Gamma$  is bounded by  $2^{p(n)}$  for some polynomial  $p(n)$ . This property, also called *polynomial expressiveness* [12], is equivalent to having either of the following two properties where *small* means of size bounded by a polynomial in the arity  $n$ :

- *small generating sets*, i.e., every relation pp-definable from  $\Gamma$  has a small subset that is not contained in any proper pp-definable subset,
- *small independent sets*, i.e., sets of tuples such that every tuple can be separated from the remaining tuples by a pp-formula, are small.

The equivalence of those properties was established in [6, Proposition 1.4].

In this paper, we study another measure of “smallness” of a constraint language, that we call *short pp-definitions*: every  $n$ -ary relation pp-definable from  $\Gamma$  is definable by some primitive positive formula of polynomial length. Examples include constraint languages encoding 2-SAT, or the consistency of linear systems over  $\mathbb{Z}_p$ .

A simple cardinality argument shows that a constraint language with short pp-definitions must have few subpowers. We conjecture that the converse is also true and thus the two properties are equivalent.

► **Conjecture 1.** *A constraint language has short pp-definitions, if and only if it has few subpowers.*

We remark that exponential-length pp-definitions are needed for constraint languages without few subpowers (cf. [6, Theorem 3.12]).

In Section 2 we give a formal definition of short pp-definitions, examples, and an exposition of related properties. An equivalent condition (definability by pp-formulas with polynomially many existential quantifiers) was studied in [23] for Boolean constraint languages (i.e., constraint languages on a two-element domain) under the name *polynomial closedness*. It can be easily seen that Conjecture 1 is true in the Boolean case, as stated in [23, Corollary 1].

In Section 4 we prove Theorem 21, the main result of our paper, which confirms the conjecture for a substantial class of constraint languages, namely those whose *polymorphism algebra* generates a *residually finite variety*. This, in particular, implies that Conjecture 1 also holds for constraint languages on three-element domains (Corollary 23). The proof

proceeds by first reducing to the case of *critical* relations (see [29]), and then employing structural theorems from universal algebra, in a similar fashion as in [9]. We explain some necessary background from universal algebra in Section 3.

Apart from being a natural property in constraint satisfaction, in Section 5 we argue that short pp-definitions have further applications in the study of the *subpower membership problem*  $\text{SMP}(\mathbf{A})$  over an algebraic structure  $\mathbf{A}$  (see [21, 24, 9]), i.e., the problem of deciding whether a given list of tuples over  $\mathbf{A}$  generates another tuple over  $\mathbf{A}$ . For algebras  $\mathbf{A}$  with few subpowers, *compact representations* provide a natural certificate for “Yes”-instances; this was used to show that  $\text{SMP}(\mathbf{A}) \in \text{NP}$  [20]. We argue that short pp-definitions can serve as a natural certificate for “No”-instances. In particular, we show how short pp-definitions impose an upper complexity bound of co-NP on the subpower membership problem. Thus, Conjecture 1 would imply that  $\text{SMP}(\mathbf{A}) \in \text{NP} \cap \text{co-NP}$ , for all algebras  $\mathbf{A}$  with few subpowers.

## 2 Preliminaries

Let  $A$  be a finite set. An  $n$ -ary relation  $R$  on  $A$  is any subset of  $n$ -tuples  $R \subseteq A^n$ . By a *constraint language* on  $A$  (its *domain*) we mean any finite set  $\Gamma = \{R_1, \dots, R_m\}$  of relations on  $A$  of arbitrary, but finite arities.

A relation  $R$  is *primitive positive definable* (or *pp-definable* for short) from  $\Gamma$ , if it is definable in first-order logic by a formula using only the relations from  $\Gamma$ , the equality relation, conjunction, and existential quantification. Equivalently, in prenex normal form:

$$R(x_1, \dots, x_n) \leftrightarrow \exists y_1 \exists y_2 \dots \exists y_k \bigwedge_{i \in \{1, \dots, C\}} S_i(z_1^i, \dots, z_{r_i}^i)$$

where  $S_i$  is an  $r_i$ -ary relational symbol representing a relation from  $\Gamma \cup \{=_A\}$  and  $z_j^i \in \{x_1, \dots, x_n, y_1, \dots, y_k\}$ . We remark that  $\text{CSP}(\Gamma)$  can be defined as the problem of deciding the primitive positive fragment of the first-order theory of  $\Gamma$ .

The set of all relations pp-definable from  $\Gamma$ , denoted by  $\langle \Gamma \rangle$ , forms a *relational clone*, i.e., a set of relations on  $A$  containing the identity relation and closed under intersections, direct products, projections, and permutations of coordinates. Any constraint language  $\Gamma$  that generates a relational clone  $\mathcal{R} = \langle \Gamma \rangle$  is called a *relational basis* of  $\mathcal{R}$ . Let us denote by  $\langle \Gamma \rangle_n$  the set of all  $n$ -ary relations pp-definable from  $\Gamma$ .

The usefulness of pp-definability for the CSPs is summarized in the following theorem going back to [19]. For a modern exposition as well as generalizations see [3] and [4].

► **Theorem 2.** *If  $\Gamma$  and  $\Delta$  are constraint languages such that  $\Delta \subseteq \langle \Gamma \rangle$ , then there is a logspace reduction from  $\text{CSP}(\Delta)$  to  $\text{CSP}(\Gamma)$ .*

In order to put Conjecture 1 on a firm footing, let us next formally define the notion of *few subpowers* [6, 16] and the central concept of the present paper, *short pp-definitions*.

► **Definition 3.** *A constraint language  $\Gamma$  has few subpowers, if there exists a polynomial  $p(n)$  such that  $|\langle \Gamma \rangle_n| \leq 2^{p(n)}$  for all  $n > 0$ .*

► **Definition 4.** *Let  $\Gamma$  be a constraint language. We say that  $\Gamma$  has:*

- pp-definitions of length [at most]  $f(n)$ , if for every  $n > 0$  and every  $R \in \langle \Gamma \rangle_n$ ,  $R$  is definable from  $\Gamma$  by a primitive positive formula  $\phi$  of length  $|\phi| \leq f(n)$ .
- short pp-definitions if  $\Gamma$  has pp-definitions of length  $p(n)$  for some polynomial  $p(n)$ .

Here we consider the length  $|\phi|$  to be simply the number of symbols in some syntactical representation of the formula. In the definition of *short pp-definitions*, one could alternatively bound the number of atomic formulas in  $\phi$ , or the number  $k$  of existentially quantified

variables by a polynomial  $p(n)$ . (The latter option was used in [23] in the notion of *polynomial closedness* of  $\langle \Gamma \rangle$ .) Note that, since  $\Gamma$  is fixed and finite, these three possible definitions coincide.

Clearly, having few subpowers is a property of the relational clone  $\langle \Gamma \rangle$ , independent of the choice of the relational basis  $\Gamma$ . We observe that the same is true for short pp-definitions. In fact, up to multiplication by a scalar, this is true for any bound  $f(n)$  on the length of pp-definitions:

► **Lemma 5.** *Let  $\Gamma$  and  $\Delta$  be constraint languages such that  $\langle \Gamma \rangle = \langle \Delta \rangle$ . If  $\Gamma$  has pp-definitions of length  $f(n)$ , then  $\Delta$  has pp-definitions of length  $O(f(n))$ . In particular,  $\Gamma$  has short pp-definitions if and only if  $\Delta$  does.*

**Proof.** Let  $R \in \langle \Delta \rangle_n = \langle \Gamma \rangle_n$ . By assumption,  $R$  has a pp-definition  $\phi_R$  from  $\Gamma$  of length at most  $f(n)$ . Since  $\Gamma \subseteq \langle \Gamma \rangle = \langle \Delta \rangle$ , every relation  $S \in \Gamma$  can be defined from  $\Delta$  by some pp-formula  $\psi_S$ . Let  $c = \max\{|\psi_S| : S \in \Gamma\}$ . If we replace every atomic formula  $S_i(z_1^i, \dots, z_{r_i}^i)$  in  $\phi_R$  by a suitable variant of the formula  $\psi_{S_i}$ , we obtain a pp-definition of  $R$  from  $\Delta$  of length at most  $c \cdot f(n)$ . ◀

Central to the algebraic approach to the CSP is the idea that constraint languages up to pp-definability (that is, relational clones) can be characterized by their *polymorphisms*. Following the terminology from [3], a  $k$ -ary operation  $f : A^k \rightarrow A$  is *compatible* with an  $n$ -ary relation  $R \subseteq A^n$ , and  $R$  is *invariant* under  $f$ , if  $f$  applied coordinate-wise to any  $k$   $n$ -tuples from  $R$  yields an  $n$ -tuple that is also in  $R$ . A *polymorphism* of a constraint language  $\Gamma$  is then any function on the domain that is compatible with all relations from  $\Gamma$ . As is usual, we write  $\text{Pol}(\Gamma)$  to denote the set of all polymorphisms of  $\Gamma$  and, similarly,  $\text{Inv}(\mathcal{F})$  for the set of all relations on the domain  $A$  invariant under a set of operations  $\mathcal{F}$ . The key connection between polymorphisms and pp-definability can be summarized in the following lemma.

► **Lemma 6** ([15, 7, 17]). *For any constraint language  $\Gamma$ ,  $\langle \Gamma \rangle = \text{Inv}(\text{Pol}(\Gamma))$ .*

Few subpowers can be characterized by the existence of an *edge polymorphism*, that is, a polymorphism satisfying certain algebraic identities (under all evaluations of variables in the domain). Such characterizations are typical in the algebraic approach to the CSP.

► **Theorem 7** ([6, 16]). *A constraint language  $\Gamma$  has few subpowers, if and only if for some  $k \geq 2$  there exists a  $k$ -edge polymorphism  $e \in \text{Pol}(\Gamma)$ , that is, a  $(k+1)$ -ary operation  $e : A^{k+1} \rightarrow A$  satisfying the following identities:*

$$\begin{aligned} e(y, y, x, x, x, \dots, x) &\approx x \\ e(y, x, y, x, x, \dots, x) &\approx x \\ e(x, x, x, y, x, \dots, x) &\approx x \\ e(x, x, x, x, y, \dots, x) &\approx x \\ &\vdots \\ e(x, x, x, x, x, \dots, y) &\approx x \end{aligned}$$

*In this case  $\text{CSP}(\Gamma) \in \text{P}$ .*

The following two special cases were important intermediate steps towards Theorem 7 (as well as the CSP dichotomy theorem) and, in particular, cover all *Boolean* (i.e., where  $A = \{0, 1\}$ ) constraint languages with few subpowers:

- A *Mal'tsev* operation is a ternary operation  $m: A^3 \rightarrow A$  satisfying the identities  $m(x, x, y) \approx m(y, x, x) \approx y$ . If  $m$  is a Mal'tsev operation, then  $e(x_1, x_2, x_3) = m(x_2, x_1, x_3)$  is a 2-edge term.
- A *near-unanimity* operation (of arity  $k \geq 3$ ) is an operation  $t$  satisfying the following identities:

$$x \approx t(y, x, \dots, x) \approx t(x, y, x, \dots, x) \approx t(x, \dots, x, y, x) \approx t(x, \dots, x, y)$$

Then  $e(x_1, x_2, \dots, x_{k+1}) = t(x_2, \dots, x_{k+1})$  is a  $k$ -edge term. A ternary near-unanimity is called a *majority*.

Let us now give two examples of constraint languages with short pp-definitions; Example 8 is invariant under a Mal'tsev operation, while Example 11 has a majority polymorphism.

► **Example 8.** The problem of checking consistency of a linear system over  $\mathbb{Z}_2$  can be encoded as CSP( $\Gamma$ ) for  $\Gamma = \{R_{\text{Lin}}, C_0, C_1\}$  where  $R_{\text{Lin}} = \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\}$  encodes “ $x_1 + x_2 = x_3$ ”,  $C_0 = \{0\}$ , and  $C_1 = \{1\}$ . Indeed, any linear equation  $x_1 + x_2 + \dots + x_n = b$  can be encoded using auxiliary variables  $y_1, \dots, y_{n-1}$  and three-variable equations

$$x_1 + x_2 = y_1, \quad y_1 + x_3 = y_2, \quad \dots, \quad y_{n-2} + x_n = y_{n-1}, \quad y_{n-1} = b$$

thus providing a pp-definition:

$$\exists y_1 \dots \exists y_{n-1} (R_{\text{Lin}}(x_1, x_2, y_1) \wedge \dots \wedge R_{\text{Lin}}(y_{n-2}, x_n, y_{n-1}) \wedge C_b(y_{n-1}))$$

The relational clone  $\langle \Gamma \rangle$  then consists of all affine subspaces of  $\mathbb{Z}_2^n$ , for any  $n > 0$ . The relations from the relational basis  $\Gamma$  are all affine and it is easy to see that relations pp-definable from affine subspaces are also affine subspaces. On the other hand, an affine subspace  $R$  of  $\mathbb{Z}_2^n$  can be described by at most  $n$  linear equations and the conjunction of the corresponding pp-formulas clearly defines  $R$ . The length of this conjunction is in  $O(n^2)$  and therefore  $\Gamma$  has not only few subpowers but also short (quadratic) pp-definitions.  $\Gamma$  is arguably one of the easiest examples of a constraint language with a Mal'tsev polymorphism; in fact  $\langle \Gamma \rangle = \text{Pol}(\{m\})$  for the Mal'tsev operation  $m(x, y, z) = x + y + z \pmod 2$ .

While Conjecture 1 is open even under the presence of a Mal'tsev polymorphism, the above example can be generalized to a *central* Mal'tsev polymorphism, that is, one which is compatible with its own function graph, i.e., the 4-ary relation  $R = \{(x, y, z, m(x, y, z)) \mid x, y, z \in A\}$ . (The reason is that the *polymorphism algebra* is then *affine*, i.e., polynomially equivalent to a module.)

For the second example, we need the following characterization of relations invariant under near-unanimity operations:

► **Theorem 9** ([2]). *If a relation  $R \subseteq A^n$  is invariant under a  $(k + 1)$ -ary near-unanimity operation  $t$ , then it is pp-definable from its projections to at most  $k$ -ary subsets of coordinates by the following formula:*

$$R(x_1, \dots, x_n) \leftrightarrow \bigwedge_{\substack{I = \{i_1, \dots, i_k\} \\ I \subseteq [n], |I| \leq k}} \text{proj}_I R(x_{i_1}, \dots, x_{i_k})$$

Here, for a relation  $R \subseteq A^n$  and a subset of coordinates  $I \subseteq \{1, 2, \dots, n\}$ , the *projection* of  $R$  to  $I = \{i_1, i_2, \dots, i_k\}$  (where  $i_1 < i_2 < \dots < i_k$ ) is the  $k$ -ary relation  $\text{proj}_I R = \{(a_{i_1}, \dots, a_{i_k}) \mid (a_1, \dots, a_n) \in R\} \in \langle \Gamma \rangle$ . Thus, if a constraint language  $\Gamma$  has a  $(k + 1)$ -ary near-unanimity polymorphism  $t$ , then every relation  $R \in \langle \Gamma \rangle_n$  can be written as the conjunction of  $\binom{n}{k}$  relations of arity  $k$  (for  $n \geq k$ ). As a direct corollary of Theorem 9 we obtain:

► **Corollary 10.** *Let  $\Gamma$  be a constraint language with a  $(k + 1)$ -ary near-unanimity polymorphism. Then  $\Gamma$  has pp-definitions of length  $O(n^k)$ .*

► **Example 11.** Corollary 10 can be exemplified by 2-SAT. The standard way to encode 2-SAT as a CSP is by using the constraint language  $\Gamma_{2\text{-SAT}} = \{R_{00}, R_{01}, R_{10}, R_{11}\}$  where  $R_{ij} = \{0, 1\}^2 \setminus \{(i, j)\}$  represent each clause type (see [3, Example 2.2]). It is well known that the relations pp-definable from  $\Gamma_{2\text{-SAT}}$  are exactly those invariant under the majority operation, i.e., the unique 3-ary near-unanimity operation on  $\{0, 1\}$ . By Corollary 10,  $\Gamma_{2\text{-SAT}}$  has quadratic pp-definitions.

► **Theorem 12** (see [23, Corollary 1]). *Let  $\Gamma$  be a Boolean constraint language with few subpowers, then  $\Gamma$  has quadratic pp-definitions. Thus Conjecture 1 holds for constraint languages  $\Gamma$  over Boolean domains.*

**Proof.** By the classification of Post's lattice, every Boolean constraint language with few subpowers has either the Mal'tsev polymorphism  $x + y + z \bmod 2$  or the (unique) majority polymorphism (this was observed, e.g., in [12]). In both cases, we obtain quadratic pp-definitions, as in Examples 8 and 11. ◀

We remark that, in general, the situation is much more complicated than in Theorem 12: Already in the 3-element case there are constraint languages that have few subpowers, but neither a Mal'tsev, nor a near-unanimity polymorphism (e.g. [8, Examples 2.1.1 and 2.1.2]).

### 3 Universal Algebra

In the following, we are going to introduce some basic notions from universal algebra that will allow us to state our main result (Theorem 21) in its full generality. In Section 3.1 we furthermore discuss how short pp-definitions behave with respect to basic algebraic constructions. For more background in universal algebra we refer to the textbooks [5, 11].

An *algebra*  $\mathbf{A} = (A; (f_i)_{i \in I}^{\mathbf{A}})$  is a first-order structure in a purely functional language  $(f_i)_{i \in I}$  (where each symbol  $f_i$  has an associated *arity*). We say  $\mathbf{A}$  is finite if its domain  $A$  is finite. A *subalgebra*  $\mathbf{B} = (B; (f_i)_{i \in I}^{\mathbf{B}})$  of an algebra  $\mathbf{A} = (A; (f_i)_{i \in I}^{\mathbf{A}})$  (denoted  $\mathbf{B} \leq \mathbf{A}$ ) is an algebra obtained by restricting all *basic operations*  $f_i^{\mathbf{A}}$  to an invariant subset  $B \subseteq A$ . The *product*  $\prod_{i \in I} \mathbf{A}_i$  of a family of algebras  $(\mathbf{A}_i)_{i \in I}$  in the same language is defined as the algebra with domain  $\prod_{i \in I} A_i$ , whose basic operations are defined coordinate-wise. A *homomorphism*  $h: \mathbf{A} \rightarrow \mathbf{B}$  between algebras is defined as a map that preserves all basic operations, i.e.,  $h(f_i^{\mathbf{A}}(a_1, \dots, a_n)) = f_i^{\mathbf{B}}(h(a_1), \dots, h(a_n))$  for all  $i \in I$ . The *kernel* of every homomorphism (i.e., the relation defined by  $(x, y) \in \theta \leftrightarrow h(x) = h(y)$ ) is a *congruence*, that is, an equivalence relation invariant under  $\mathbf{A}$ . Conversely, for every congruence  $\alpha$  of  $\mathbf{A}$ , it is easy to see, that one can construct a *quotient algebra*  $\mathbf{A}/\alpha$ , as the homomorphic image of the quotient mapping  $x \mapsto x/\alpha$ . Under the inclusion order, the set of all congruence of an algebra  $\mathbf{A}$  forms the *congruence lattice*  $\text{Con}(\mathbf{A})$ . The minimal element of this lattice is always the trivial congruence  $0_{\mathbf{A}} = \{(x, x) \mid x \in A\}$ . An algebra  $\mathbf{A}$  is called *subdirectly irreducible* if  $0_{\mathbf{A}}$  has a unique cover in  $\text{Con}(\mathbf{A})$ , i.e., there is a unique minimal non-trivial congruence.

By H, S, and P we denote the closure of a set of algebras under homomorphic images, subalgebras, and products respectively. It is well-known that the closure of any set of algebras under HSP is a *variety*, i.e., a class of algebras defined by a set of identities (by Birkhoff's theorem, see, e.g., [5]). A variety is called *residually finite* if (up to isomorphism) it only contains finitely many subdirectly irreducible algebras, all of which are finite.

### 3.1 Algebras with short pp-definitions

If we assign a function symbol to every element of  $\text{Pol}(\Gamma)$ , for a constraint language  $\Gamma$ , then we can also regard  $\text{Pol}(\Gamma)$  as an algebra (the *polymorphism algebra* of  $\Gamma$ ). On the other hand, for every algebra  $\mathbf{A}$ , its invariant relations  $\text{Inv}(\mathbf{A})$  form a relational clone. Thus, it makes sense to say that a finite algebra  $\mathbf{A}$  has *few subpowers* if  $\text{Inv}(\mathbf{A})$  has few subpowers.

Note that a relation  $R$  is invariant under  $\mathbf{A}$  if and only if  $R \leq \mathbf{A}^n$  for some  $n$ , i.e.,  $R$  is a subalgebra of some power of  $\mathbf{A}$  (such  $R$  is also called a *subpower of  $\mathbf{A}$* , which motivates the notion of having “few subpowers”).

By a (non-constructive) proof of Aichinger, Mayr and McKenzie [1], for every algebra  $\mathbf{A}$  with few subpowers there exists a finite relational basis  $\Gamma$  of  $\text{Inv}(\mathbf{A})$ , i.e., a constraint language, such that  $\text{Inv}(\mathbf{A}) = \langle \Gamma \rangle$  (for general algebras  $\mathbf{A}$  this is not the case). Thus, it makes sense to define the following:

► **Definition 13.** *An algebra  $\mathbf{A}$  has pp-definitions of length  $f(n)$ , if there exists a constraint language  $\Gamma$  such that  $\text{Inv}(\mathbf{A}) = \langle \Gamma \rangle$ , and  $\Gamma$  has pp-definitions of length  $f(n)$ . An algebra  $\mathbf{A}$  has short pp-definitions, if it has pp-definitions of length  $p(n)$ , for some polynomial  $p$ .*

Note that, by Lemma 5, having short pp-definitions is independent of the choice of the relational basis  $\Gamma$ . By the following lemma, having short pp-definitions is also preserved under forming finite powers of algebras. The proof is provided in Section A.1 in the Appendix.

► **Lemma 14.** *Let  $\mathbf{A}$  be an algebra and  $\mathbf{B} = \mathbf{A}^k$  for some  $k > 1$ . Then  $\mathbf{B}$  has pp-definitions of length  $O(f(n))$  if and only if  $\mathbf{A}$  has pp-definitions of length  $O(f(\lceil \frac{n}{k} \rceil))$ .*

In the following, we will also work with multi-sorted relations, as this provides a natural framework for our proof in Section 4. More specifically, if  $\mathcal{A}$  is a finite set of finite algebras of the same language, then it still makes to study the set of all invariant relations  $R \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  for  $\mathbf{A}_1, \dots, \mathbf{A}_n \in \mathcal{A}$ . In particular, the set of all such relations will still form a relational clone (where variables have possibly different domains  $A_i$ ). If, furthermore, all elements of  $\mathcal{A}$  have few subpowers, the finite relational basis result of Aichinger, Mayr and McKenzie [1] still applies, and it makes sense to define the property of having short pp-definitions for  $\mathcal{A}$  (we refrain from giving technical details here). We remark that studying constraint languages in which the variables can come from different domains is a fairly standard viewpoint in CSP; it was, for example, used in the proof of the CSP dichotomy theorem by Zhuk [28].

This multisorted approach allows us to consider relations over the closure  $\text{HS}(\mathbf{A})$  of  $\mathbf{A}$  under homomorphic images and subalgebras instead of only  $\mathbf{A}$  itself. This is justified by the following lemma; the proof is provided in Section A.2 of the Appendix.

► **Lemma 15.** *An algebra  $\mathbf{A}$  has pp-definitions of length  $O(f(n))$ , if and only if the family of algebras  $\text{HS}(\mathbf{A})$  has (multisorted) pp-definitions of length  $O(f(n))$ .*

In particular, Lemma 15 implies that  $\mathbf{A}$  has short pp-definitions, if and only if  $\text{HS}(\mathbf{A})$  has (multisorted) short pp-definitions. We remark that Lemma 15 does *not* imply that any *single* algebra  $\mathbf{B} \in \text{HS}(\mathbf{A})$  has short pp-definitions if  $\mathbf{A}$  does. In fact, we do not know if this is true (see Question 25 in the Discussion section).

## 4 Main result

In this section, we prove the main result of our paper. We first need to introduce some standard definitions that found prominent use in the universal algebraic approach to constraint satisfaction before (see, e.g., [8, Chapter 2]).

A relation  $R \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  is called *critical* if it is  $\wedge$ -irreducible, i.e., it cannot be written as the intersection of strictly bigger relations  $Q \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$ , and it has *no dummy variables*, i.e., it depends on all of its inputs.

A binary relation  $R \subseteq A \times B$  has the *parallelogram property* if  $(a, c), (a, d), (b, c) \in R$  implies  $(b, d) \in R$ . An  $n$ -ary relation  $R \subseteq A_1 \times A_2 \times \dots \times A_n$  has the *parallelogram property*, if for all subsets  $I \subset \{1, 2, \dots, n\}$  it has the parallelogram property when considered as a binary relation  $R \subseteq (\prod_{i \in I} A_i) \times (\prod_{j \notin I} A_j)$ . The *signature* of  $R$  is the following set of triples:

$$\text{Sig}(R) = \{(i, a, b) \in [n] \times A_i^2 \mid \exists \bar{x}, \bar{y} \in R \text{ with } x_j = y_j \text{ for } j = 1, \dots, i-1 \text{ and } x_i = a, y_i = b\}$$

In the proof of Lemma 17 below, we will need the following straightforward observation.

► **Observation 16.** *If  $R$  has the parallelogram property,  $R \subseteq S$ , and  $\text{Sig}(R) = \text{Sig}(S)$ , then  $R = S$ .*

Using these notions, we can reduce the problem of finding short pp-definitions to critical relations with the parallelogram property:

► **Lemma 17.** *Let  $\mathbf{A}$  be an algebra with a  $k$ -edge term. If all the critical relations  $R \leq \mathbf{A}^n$  with parallelogram property have pp-definitions of length at most  $f(n)$ , then  $\mathbf{A}$  has pp-definitions of length  $O(n^k + n \cdot f(n))$ .*

**Proof.** Clearly, every relation  $R \leq \mathbf{A}^n$  is the intersection of the  $\wedge$ -irreducible relations above it (in the inclusion order), thus it can be written as a conjunction of critical relations. Hence we only need to give an upper bound on the number of such critical relations.

By [20, Theorem 3.6], the presence of a  $k$ -edge term implies that every critical relation is either of arity  $\leq k$ , or has the parallelogram property. This further implies (see, e.g., [8, Corollary 2.3.5.]) that  $R = R' \wedge \bigwedge_{I \subseteq [n], |I| \leq k} \text{proj}_I(R)$ , where  $R' \leq \mathbf{A}^n$  is the minimal invariant relation containing  $R$  and having the parallelogram property. Clearly,  $\bigwedge_{I \subseteq [n], |I| \leq k} \text{proj}_I(R)$  can be written as the conjunction of at most  $c \cdot n^k$  many critical relations, for some  $c > 0$ .

The relation  $R'$ , if not already  $\wedge$ -irreducible itself, is given by the intersection of all  $\wedge$ -irreducible relations  $S > R'$  that have the parallelogram property. Denote by  $\mathcal{S}$  the set of all such relations. For every  $(i, a, b) \notin \text{Sig}(R')$  where  $a \in \text{proj}_i R'$  choose  $S_{(i,a,b)} \in \mathcal{S}$  such that  $(i, a, b) \notin \text{Sig}(S_{(i,a,b)})$ . Then by Observation 16,  $R' = \bigcap_{(i,a,b) \notin \text{Sig}(R')} S_{(i,a,b)}$  which is an intersection of at most  $n \cdot |A|^2$   $\wedge$ -irreducible relations. (To see that such  $S_{(i,a,b)}$  exists, let  $\bar{x} \in R'$  be such that  $x_i = a$  and let  $\bar{y}$  be such that  $y_i = b, y_j = x_j$  for  $j \neq i$ . We can choose  $S_{(i,a,b)}$  to be a maximal relation containing  $R$  but omitting  $\bar{y}$ .)

Consequently,  $R'$  can be defined as a conjunction of at most  $n \cdot |A|^2$  many critical relations with the parallelogram property which concludes the proof. ◀

We remark that an analogous statement to Lemma 17 also holds for multisorted relations  $R \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$ , such that the sorts  $\mathbf{A}_i$  come from a finite set of algebras that have a common  $k$ -edge term. In particular, this is the case for  $\mathbf{A}_i \in \text{HS}(\mathbf{A})$ , if  $\mathbf{A}$  has a  $k$ -edge term (cf. Lemma 15).

When dealing with multisorted relations  $R$  over  $\text{HS}(\mathbf{A})$ , we can furthermore always restrict the domain  $\mathbf{A}_i$  of the  $i$ -th variable of a relation  $R$  to its projection  $\text{proj}_i(R) \leq \mathbf{A}_i$ . So, without loss of generality, we can assume that  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  is *subdirect*, i.e., its projection to every coordinate  $i$  is the full domain  $\mathbf{A}_i$ .

For a subdirect relation  $R \leq_{sd} \mathbf{B} \times \mathbf{C}$  with the parallelogram property, let us define the *linkedness congruence*  $\theta_B$  on  $\mathbf{B}$  by  $(x, y) \in \theta_B \leftrightarrow (\exists c \in C)(R(x, c) \wedge R(y, c))$ . For a general relation  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with the parallelogram property, and any proper subset  $I \subset [n]$

of coordinates, we define the *linkedness congruence*  $\theta_I$  on  $\text{proj}_I(R)$  analogously, where we consider  $R$  as a binary relation between  $\text{proj}_I(R)$  and  $\text{proj}_{[n]\setminus I}(R)$  (we write  $\theta_i$  instead of  $\theta_{\{i\}}$ ). It follows from the parallelogram property that  $\theta_I$  is indeed a congruence of  $\text{proj}_I(R)$ . A subdirect relation  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  is called *reduced* if every tuple  $(a_1, \dots, a_n) \in R$  is already uniquely determined by  $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ , for any coordinate  $i$ ; in other words,  $\theta_i$  is trivial, for every  $i = 1, \dots, n$ . By the following lemma, we can reduce the quest for short pp-definitions to *reduced*, subdirect, critical relations with the parallelogram property:

► **Lemma 18.** *Let  $\mathbf{A}$  be an algebra with a  $k$ -edge term. Assume that all relations  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with  $\mathbf{A}_i \in \text{HS}(\mathbf{A})$  that are reduced, critical, and have the parallelogram property, have (multisorted) pp-definitions of length at most  $f(n)$ . Then  $\mathbf{A}$  has pp-definitions of length  $O(n^k + n \cdot f(n))$ .*

**Proof.** We first prove that we can pp-define all critical relations  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with  $\mathbf{A}_i \in \text{HS}(\mathbf{A})$  that have the parallelogram property (but are not necessarily reduced), by pp-definitions of length at most  $O(f(n))$ . Given such a relation, let us consider the linkedness congruence  $\theta_i \in \text{Con}(\mathbf{A}_i)$  for every coordinate  $i$ . Then let us define the quotient  $R' = R/(\theta_1, \dots, \theta_n)$ . It is not hard to see that  $R' \leq_{sd} \mathbf{A}_1/\theta_1 \times \dots \times \mathbf{A}_n/\theta_n$ , is also critical, and has the parallelogram property. Furthermore, by definition of  $\theta_i$ ,  $R'$  is reduced.

Since  $R$  has the parallelogram property, it is equal to the full preimage of  $R'$  under the quotient map  $(x_1, \dots, x_n) \mapsto (x_1/\theta_1, \dots, x_n/\theta_n)$ . Thus, every pp-definition  $\phi'(x_1, \dots, x_n)$  of  $R'$  gives rise to the pp-definition  $\exists y_1, \dots, y_n (\bigwedge_{i=1}^n (x_i/\theta_i = y_i) \wedge \phi'(y_1, \dots, y_n))$  of length  $O(f(n))$  which defines  $R$ ; this proves our claim. The statement of the lemma then follows directly from (the multi-sorted version of) Lemma 17, and Lemma 15. ◀

Any relation  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  as in Lemma 18 comes with several nice properties (in algebraic terms, it is a *graph of a joint similarity* between the algebras  $\mathbf{A}_i$ , cf. [8, Section 2.3.1]). We are mainly going to need the following property in Lemma 19, respectively its generalization in Lemma 20:

► **Lemma 19** ([20, Lemma 2.4]). *Let  $\mathbf{A}_1, \dots, \mathbf{A}_n$  be algebras with few subpowers, and let  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  be a reduced, critical relation with the parallelogram property. Then every  $\mathbf{A}_i$  is subdirectly irreducible.*

► **Lemma 20.** *Let  $\mathbf{A}_1, \dots, \mathbf{A}_n$  be algebras with few subpowers, and let  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  be a critical relation with the parallelogram property. For  $I \subset [n]$ , let  $\theta_I$  be the linkedness congruence on  $\text{proj}_I(R)$  with respect to  $R$ . Then  $\theta_I$  is  $\wedge$ -irreducible.*

**Proof.** Since  $R$  is  $\wedge$ -irreducible, there is a unique cover  $R^* > R$  in the lattice of all subalgebras of  $\mathbf{A}_1 \times \dots \times \mathbf{A}_n$ . A tuple  $\bar{a} = (a_1, \dots, a_n) \in R^* \setminus R$  is called a *key tuple* of  $R$  (cf. [29]). It follows from the criticality of  $R$  that for every  $j = 1, \dots, n$ , there is a tuple  $(a_1, \dots, b_j, \dots, a_n) \in R$  that only differs from  $\bar{a}$  at position  $j$ .

For simplicity, let us assume that  $I = \{1, 2, \dots, i\}$  with  $i < n$ . Then, the linkedness-congruence  $\theta_I$  has an equivalence class containing all elements of the form  $(a_1, \dots, b_j, \dots, a_i)$  for  $j = 1, \dots, i$ . Note that  $(a_1, a_2, \dots, a_i) \in \text{proj}_I(R)$  is not an element of this class.

To prove that  $\theta_I$  is  $\wedge$ -irreducible, let  $\theta'$  be a congruence strictly above  $\theta_I$ . We claim that then  $\theta'$  must also contain the pair  $((a_1, a_2, \dots, a_i), (b_1, a_2, \dots, a_i))$ . To prove the claim, let us define  $R'(\bar{x}) = \exists \bar{y}_I (\theta'(\bar{x}_I, \bar{y}_I) \wedge R(\bar{y}_I, \bar{x}_{[n]\setminus I}))$ . As  $R'$  properly contains  $R$ , it also must contain its cover  $R^*$ , and thus the key tuple  $(a_1, a_2, \dots, a_n)$ . Moreover, the linkedness congruence of  $R'$  on coordinates  $I$  is equal to  $\theta'$ , thus  $\theta'$  must contain the pair  $((a_1, a_2, \dots, a_i), (b_1, a_2, \dots, a_i))$ . So  $\theta_I$  has a unique cover  $\theta_I^*$ , which is the congruence generated by  $\theta_I \cup \{((a_1, a_2, \dots, a_i), (b_1, a_2, \dots, a_i))\}$ ; this finishes the proof. ◀

We are now ready to prove our main result:

► **Theorem 21.** *Let  $\mathbf{A}$  be an algebra with a  $k$ -edge term, and assume that  $\text{HSP}(\mathbf{A})$  is residually finite. Then  $\mathbf{A}$  has pp-definitions of length  $O(n^k)$ .*

**Proof.** Let  $\mathcal{V}_{SI}$  consist of all subdirectly irreducible elements of  $\text{HSP}(\mathbf{A})$ . Since  $\text{HSP}(\mathbf{A})$  is residually finite,  $\mathcal{V}_{SI}$  contains up to isomorphism only finitely many algebras, all of which are finite. In particular  $\mathcal{V}_{SI}$  is a subset of  $\text{HS}(\mathbf{A}^l)$ , for some finite power  $l$ .

By Lemma 14, it is enough to prove pp-definitions of length  $O(n^k)$  for  $\mathbf{A}^l$ . By Lemma 18, it suffices to prove that every reduced, critical relation  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with  $\mathbf{A}_i \in \text{HS}(\mathbf{A}^l)$  that has the parallelogram property, has a (multisorted) pp-definition of linear length.

Let  $\Gamma$  be the set of all at most ternary invariant relations over  $\text{HS}(\mathbf{A}^l)$ . We construct a pp-definition of  $R$  from  $\Gamma$  of length linear in  $n$ , by induction on  $n$ . For  $n \leq 3$ ,  $R$  itself is in  $\Gamma$ .

For general  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with the parallelogram property, recall the definition of the linkedness congruence  $\theta_I$ . We then define the algebra  $\mathbf{A}_{1,2} = \text{proj}_{\{1,2\}}(R)/\theta_{\{1,2\}}$ , and the relations  $Q = \{(x_1, x_2, y_{1,2}) \in A_1 \times A_2 \times A_{1,2} \mid y_{1,2} = (x_1, x_2)/\theta_{\{1,2\}}\}$  and  $R' = \{(y_{1,2}, x_3, \dots, x_n) \mid \exists x_1, x_2 (Q(x_1, x_2, y_{1,2}) \wedge R(x_1, x_2, x_3, \dots, x_n))\}$ . Note that  $Q \leq \mathbf{A}_1 \times \mathbf{A}_2 \times \mathbf{A}_{1,2}$ , and  $R' \leq \mathbf{A}_{1,2} \times \mathbf{A}_3 \times \dots \times \mathbf{A}_n$ . Since  $R$  has the parallelogram property,  $R$  can be defined by the pp-formula  $(\exists y_{1,2} \in \mathbf{A}_{1,2}) (Q(x_1, x_2, y_{1,2}) \wedge R'(y_{1,2}, x_3, \dots, x_n))$ .

By Lemma 20,  $\theta_{\{1,2\}}$  is  $\wedge$ -irreducible. This implies that  $\mathbf{A}_{1,2}$  is subdirectly irreducible and hence an element of  $\mathcal{V}_{SI} \subseteq \text{HS}(\mathbf{A}^l)$ . In particular, this means that  $Q$  is a relation from our relational basis  $\Gamma$ . The relation  $R'$  is of arity  $n - 1$ , and thus, by induction assumption, has a pp-definition of linear length. This finishes our proof. ◀

Note that, although in the proof of Theorem 21 we found a *ternary* constraint language  $\Gamma$  defining the reduced critical relations  $R \leq_{sd} \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with parallelogram property, the same may not be true for the original algebra  $\mathbf{A}$ . An explicit bound on the maximal required arity is given by  $3l$ , where  $l$  is such that all subdirectly irreducible elements of  $\text{HSP}(\mathbf{A})$  are contained in  $\text{HS}(\mathbf{A}^l)$ . We are not aware of any better bound than the double exponential  $l \leq |A|^{|A|^{|A|+1}+1}$  [14] (see also [8, Theorem A.5.27.]).

As an immediate consequence of Theorem 21 we get that every 3-element algebra with few subpowers has short pp-definitions, confirming Conjecture 1 for the 3-element case. It is well known that few subpowers imply *congruence modularity* [6, Theorem 4.2]; thus we can use the following fact:

► **Theorem 22** ([8, Corollary A.5.31.]). *Let  $\mathbf{A}$  be an algebra on a 3-element set, such that  $\text{HSP}(\mathbf{A})$  is congruence modular. Then  $\text{HSP}(\mathbf{A})$  is residually finite.*

► **Corollary 23.** *Let  $\Gamma$  be a constraint language on a 3-element domain. Then  $\Gamma$  has short pp-definitions if and only if  $\Gamma$  has few subpowers. More precisely,  $\Gamma$  has pp-definitions of length  $O(n^k)$ , where  $k$  is the minimal number such that  $\Gamma$  has a  $k$ -edge polymorphism.*

**Proof.** Let us assume that  $\Gamma$  is a constraint language with a  $k$ -edge polymorphism, and let  $\mathbf{A} = \text{Pol}(\Gamma)$  be its polymorphism algebra. Since the existence of an edge operation implies that  $\text{HSP}(\mathbf{A})$  is congruence modular [6, Theorem 4.2], by Theorem 22,  $\text{HSP}(\mathbf{A})$  is residually finite. By Theorem 21,  $\mathbf{A}$ , and thus also  $\Gamma$ , has pp-definitions of length  $O(n^k)$ .

If  $\Gamma$  has few subpowers, then by Theorem 7, it has a  $k$ -edge polymorphism for some  $k$ . Thus  $\Gamma$  has short pp-definitions if it has few subpowers. ◀

## 5 The Subpower Membership Problem

The *Subpower Membership Problem*  $\text{SMP}(\mathbf{A})$  of a finite algebra  $\mathbf{A}$  is the computational problem in which the input consists of a list of tuples  $\bar{b}, \bar{a}_1, \dots, \bar{a}_k \in A^n$ , for arbitrary  $n \geq 1$ , and one needs to decide whether  $\bar{b}$  lies in the subalgebra  $\text{Sg}_{\mathbf{A}^n}(\bar{a}_1, \dots, \bar{a}_k)$  generated by  $\bar{a}_1, \dots, \bar{a}_k$ , i.e., in the smallest  $R \leq \mathbf{A}^n$  that contains  $\bar{a}_1, \dots, \bar{a}_k$ .

The existence of an efficient algorithm for  $\text{SMP}(\mathbf{A})$  implies that it is feasible to represent the relations in  $\text{Inv}(\mathbf{A})$  by some generating set of tuples. In particular, in the context of constraint satisfaction, it was remarked by several authors (see, e.g., [9]) that a polynomial-time algorithm for  $\text{SMP}(\mathbf{A})$  would allow us to define constraint satisfaction problems over *infinite* constraint languages  $\Gamma \subseteq \text{Inv}(\mathbf{A})$ , where the constraint relations in  $\Gamma$  are encoded by generating tuples. In [16], any algebra  $\mathbf{A}$  with  $\text{SMP}(\mathbf{A})$  in P was referred to as *polynomially evaluable*.

While there are algebras for which  $\text{SMP}(\mathbf{A})$  is EXPTIME-complete [22], it was asked in [16, Question 3] whether all algebras with few subpowers are polynomially evaluable. An affirmative answer was given for several special cases [24, 9], but the question still remains open in general. The best general upper bound on the complexity of  $\text{SMP}(\mathbf{A})$  for algebras with few subpowers is NP [9]. This bound is based on the fact that membership of an element in a relation  $R \leq \mathbf{A}^n$  can always be witnessed by a *compact representation* of  $R$ , i.e., a small, canonical generating set. The difficulty in finding *deterministic* polynomial algorithms lies in efficiently computing such compact representations of  $R$  from an arbitrary generating set.

Note that for an algebra  $\mathbf{A}$  with  $\text{Inv}(\mathbf{A}) = \langle \Gamma \rangle$ , the *non-membership* of a tuple  $\bar{b}$  in a relation  $\text{Sg}_{\mathbf{A}^n}(\bar{a}_1, \dots, \bar{a}_k)$  can be witnessed by a pp-formula  $\phi(\bar{x})$  over  $\Gamma$ , such that  $\phi$  holds for all tuples  $\bar{a}_1, \dots, \bar{a}_k$ , but not for  $\bar{b}$ .

If  $\Gamma$  has short pp-definitions, we can guess such a certificate  $\phi$  for “No”-instances of  $\text{SMP}(\mathbf{A})$ , and verify it in polynomial time. As a direct consequence of this (and the fact that short pp-definitions imply few subpowers), we obtain the following:

► **Theorem 24.** *Let  $\mathbf{A}$  be an algebra with short pp-definitions. Then  $\text{SMP}(\mathbf{A}) \in \text{NP} \cap \text{co-NP}$ .*

In particular, Conjecture 1 would imply that  $\text{SMP}(\mathbf{A}) \in \text{NP} \cap \text{co-NP}$  for every algebra  $\mathbf{A}$  with few subpowers. Note, however, that in the setting of our main result (Theorem 21), this does not provide any progress on the subpower membership problem, since it was shown in [9] that  $\text{SMP}(\mathbf{A})$  is even in P for every algebra  $\mathbf{A}$  with few subpowers that generates a residually finite variety.

## 6 Discussion

By Theorem 21, constraint languages  $\Gamma$  with few subpowers, whose polymorphism algebra generates a residually finite variety, have short pp-definitions. While this confirms Conjecture 1 for a large subclass of constraint languages, much work remains to prove the conjecture in full generality.

The condition of residual finiteness does not bear much importance in constraint satisfaction, it is mainly used in purely algebraic contexts. Important steps to connect short pp-definitions closer to the theory of constraint satisfaction would be to extend our results to specific tractability classes (such as constraint languages with Mal'tsev polymorphisms), and to show invariance under *pp-interpretations*:

► **Question 25.** *Let  $\Gamma$  and  $\Delta$  be two constraint languages, such that  $\Delta$  is pp-interpretable in  $\Gamma$  and  $\Gamma$  has short pp-definitions. Then, does  $\Delta$  also have short pp-definitions?*

Pp-interpretations are a generalization of pp-definitions, that describe certain gadget reductions between constraint languages on different domains. All standard tractability classes (including few subpowers, Mal'tsev, near-unanimity) are closed under pp-interpretations, which motivates Question 25 (note that Conjecture 1 implies a positive answer). We remark that we do not know the answer to this question even for  $\Delta$  being *pp-definable* in  $\Gamma$  (as Lemma 5 assumes *pp-interdefinability*). In the special case of *pp bi-interpretable* structures, Question 25 has a positive answer by a straightforward generalization of the proof of Lemma 5, we thank the anonymous reviewer for this observation. In algebraic terms, Question 25 asks whether for  $\mathbf{A}$  with short pp-definitions, it is the case that also every *extension* of every algebra  $\mathbf{B} \in \text{HSP}^{\text{fin}}(\mathbf{A})$  has short pp-definitions; for finite powers  $\mathbf{P}^{\text{fin}}$ , we verified the statement in Lemma 14.

As discussed in Section 5, it is also essential for progress on the Subpower Membership Problem to extend our results to algebras  $\mathbf{A}$  that do not generate residually finite varieties. While we did not present any results on this in this paper, we are aware of singular examples of such algebras with short pp-definitions (such as the 4-element algebra, described by Brady in [8, Example 2.3.2.]; short pp-definitions follow directly from his analysis).

In order to improve the complexity result of Theorem 24 and put  $\text{SMP}(\mathbf{A})$  in the class P, we would need an explicit method of efficiently computing a short pp-definition for a relation  $R = \text{Sg}_{\mathbf{A}^n}(\bar{a}_1, \dots, \bar{a}_k)$  given by its generators  $\bar{a}_1, \dots, \bar{a}_k$ . This motivates the following question:

► **Question 26.** *Let  $\Gamma$  be a constraint language with short pp-definitions. Is there a polynomial-time algorithm that computes a (short) pp-definition of a relation  $R \leq \langle \Gamma \rangle_n$ , given by a set of generators  $\bar{a}_1, \dots, \bar{a}_k$ ?*

We remark that over Boolean domains, Question 26 has a positive answer (see Examples 8 and 11).

Finally, recall that the bound from Theorem 21 is a polynomial of degree  $k$  if  $\Gamma$  has a  $k$ -edge polymorphism. It is therefore tempting to conjecture that the same degree could be enough in general for Conjecture 1. Note that the number of  $n$ -ary pp-definable relations, for  $\Gamma$  with a  $k$ -edge polymorphism, is known to be in  $2^{O(n^k)}$  [16, Theorem 3.4].

---

## References

- 1 Erhard Aichinger, Peter Mayr, and Ralph McKenzie. On the number of finite algebraic structures. *Journal of the European Mathematical Society*, 16(8):1673–1686, September 2014. doi:10.4171/jems/472.
- 2 Kirby A. Baker and Alden F. Pixley. Polynomial interpolation and the Chinese Remainder Theorem for algebraic systems. *Mathematische Zeitschrift*, 143(2):165–174, June 1975. doi:10.1007/BF01187059.
- 3 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and How to Use Them. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.
- 4 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, 2018. doi:10.1007/s11856-017-1621-9.
- 5 Clifford Bergman. *Universal Algebra: Fundamentals and Selected Topics*. Chapman and Hall/CRC, New York, November 2011. doi:10.1201/9781439851302.
- 6 Joel Berman, Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, 362(3):1445–1473, March 2010. doi:10.1090/S0002-9947-09-04874-0.

- 7 V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras, part I and II. *Cybernetics*, 5:243–539, 1969.
- 8 Zarathustra Brady. Notes on CSPs and Polymorphisms, October 2022. arXiv:2210.07383 [cs, math]. doi:10.48550/arXiv.2210.07383.
- 9 Andrei Bulatov, Peter Mayr, and Ágnes Szendrei. The Subpower Membership Problem for Finite Algebras with Cube Terms. *Logical Methods in Computer Science*, Volume 15, Issue 1, February 2019. doi:10.23638/LMCS-15(1:11)2019.
- 10 Andrei A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, October 2017. doi:10.1109/FOCS.2017.37.
- 11 S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer New York, November 1981.
- 12 Hubie Chen. The expressive rate of constraints. *Annals of Mathematics and Artificial Intelligence*, 44(4):341–352, August 2005. doi:10.1007/s10472-005-7031-4.
- 13 Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, January 1998. doi:10.1137/S0097539794266766.
- 14 Ralph Freese and Ralph McKenzie. *Commutator Theory for Congruence Modular Varieties*. CUP Archive, August 1987.
- 15 David Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.
- 16 Paweł Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and Learnability Arising from Algebras with Few Subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, January 2010. doi:10.1137/090775646.
- 17 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1):185–204, June 1998. doi:10.1016/S0304-3975(97)00230-2.
- 18 Peter Jeavons, David Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, May 1998.
- 19 Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, July 1997. doi:10.1145/263867.263489.
- 20 Keith A. Kearnes and Ágnes Szendrei. Clones of algebras with parallelogram terms. *International Journal of Algebra and Computation*, 22(01):1250005, February 2012. doi:10.1142/S0218196711006716.
- 21 Dexter Kozen. Complexity of finitely presented algebras. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, pages 164–177, New York, NY, USA, May 1977. Association for Computing Machinery. doi:10.1145/800105.803406.
- 22 Marcin Kozik. A finite set of functions with an EXPTIME-complete composition problem. *Theoretical Computer Science*, 407(1):330–341, November 2008. doi:10.1016/j.tcs.2008.06.057.
- 23 Victor Lagerkvist and Magnus Wahlström. Polynomially Closed Co-clones. In *2014 IEEE 44th International Symposium on Multiple-Valued Logic*, pages 85–90, May 2014. doi:10.1109/ISMVL.2014.23.
- 24 Peter Mayr. The subpower membership problem for Mal'cev algebras. *International Journal of Algebra and Computation*, 22(07):1250075, November 2012. doi:10.1142/S0218196712500750.
- 25 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 216–226, New York, NY, USA, May 1978. Association for Computing Machinery. doi:10.1145/800133.804350.
- 26 Charles C. Sims. Computational methods in the study of permutation groups. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, January 1970. doi:10.1016/B978-0-08-012975-4.50020-5.

- 27 Dmitriy Zhuk. A Proof of CSP Dichotomy Conjecture. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 331–342, October 2017. doi:10.1109/FOCS.2017.38.
- 28 Dmitriy Zhuk. A Proof of the CSP Dichotomy Conjecture. *Journal of the ACM*, 67(5):30:1–30:78, August 2020. doi:10.1145/3402029.
- 29 Dmitriy N. Zhuk. Key (critical) relations preserved by a weak near-unanimity function. *Algebra universalis*, 77(2):191–235, April 2017. doi:10.1007/s00012-017-0426-3.

## A Omitted proofs of technical lemmata

### A.1 Proof of Lemma 14

**Proof.** It is straightforward to see that any relation  $R \subseteq B^n$  is invariant under  $\mathbf{B}$  if and only if it is invariant under  $\mathbf{A}$ , when interpreted as an  $kn$ -ary relation on  $A$ .

We are first going to prove the “only if” direction. Let  $\Delta$  be a relational basis of  $\text{Inv}(\mathbf{B})$ . By interpreting every  $m$ -ary relation  $Q \in \Delta$  as a  $km$ -ary relation  $Q' \leq \mathbf{A}^{km}$ , we obtain a relational basis  $\Delta' = \{Q' \mid Q \in \Delta\}$  of  $\text{Inv}(\mathbf{A})$ .

Let  $R \leq \mathbf{A}^n$ . By adding dummy variables, we can assume without loss of generality that  $n = k\ell$  for  $\ell = \lceil \frac{n}{k} \rceil$ . Then by assumption,  $R$  considered as an  $\ell$ -ary relation over  $B$  has a pp-definition  $\phi(x_1, \dots, x_\ell)$  from  $\Delta$  of length in  $O(f(\ell))$ . If we substitute each ( $B$ -valued) variable in  $\phi$  by a  $k$ -tuple of ( $A$ -valued) variables, and each  $\Delta$ -predicate  $Q$  in  $\phi$  by the corresponding  $\Delta'$ -predicate  $Q'$ , then we obtain a pp-definition  $\phi'$  of  $R$  of length at most  $k \cdot f(\ell) = k \cdot f(\lceil \frac{n}{k} \rceil)$ . Note further that existentially quantifying all the additionally added dummy variables adds only constantly many symbols. Thus  $\mathbf{A}$  has pp-definitions of length  $O(k \cdot f(\lceil \frac{n}{k} \rceil)) = O(f(\lceil \frac{n}{k} \rceil))$ .

Now let us prove the “if” direction. Let  $\Gamma$  be a relational basis of  $\text{Inv}(\mathbf{A})$ . Let  $e: A \rightarrow A^k$  be the map  $x \mapsto (x, \dots, x)$ . For every  $Q \in \Gamma$ , we define  $Q' = \{(e(x_1), \dots, e(x_m)) \mid (x_1, \dots, x_m) \in Q\} \leq \mathbf{B}^m$ , and we define the binary relations  $P_i = \{((x_1, \dots, x_k), e(x_i)) \mid (x_1, \dots, x_k) \in B\} \leq \mathbf{B}^2$ , for  $i = 1, \dots, k$ . We construct the relational basis of  $\text{Inv}(\mathbf{B})$  as  $\Gamma' = \{Q' \mid Q \in \Gamma\} \cup \{P_1, \dots, P_k\}$ .

Let  $R' \leq \mathbf{B}^n$  and let  $R \leq \mathbf{A}^{kn}$  be the relation  $R'$  considered as a  $kn$ -ary relation over  $A$ . By assumption, there exists a pp-definition  $\phi(x_1, \dots, x_{nk})$  of  $R \leq \mathbf{A}^{kn}$  over  $\Gamma$  of length in  $O(f(\lceil \frac{kn}{k} \rceil)) = O(f(n))$ . Let  $z_1, \dots, z_\ell$  be its existentially quantified variables. Let us then define  $\phi'(y_1, y_2, \dots, y_n)$  as a pp-formula over  $\Gamma'$  with existentially quantified variables  $x'_1, \dots, x'_{nk}, z'_1, \dots, z'_\ell$  and predicates  $P_i(y_j, x'_{(j-1)(k+i)})$  for all  $i \in [k], j \in [n]$ , as well as  $Q'(u'_1, \dots, u'_m)$  for every predicate  $Q(u_1, \dots, u_m)$  in  $\phi$  with  $u_i \in \{x_1, \dots, x_{nk}, z_1, \dots, z_\ell\}$ . It is easy to check that  $\phi'$  defines  $R' \leq \mathbf{B}^n$  over  $\Gamma'$ . Clearly, the length of  $\phi'$  is in  $O(f(n))$ . ◀

### A.2 Proof of Lemma 15

**Proof.** Let  $\Gamma$  be a relational basis of  $\text{Inv}(\mathbf{A})$ , and let  $R \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with  $\mathbf{A}_i \in \text{HS}(\mathbf{A})$ . So  $\mathbf{A}_i = h_i(\mathbf{S}_i)$  for a subalgebra  $\mathbf{S}_i \leq \mathbf{A}$  and a homomorphism  $h_i: \mathbf{S}_i \rightarrow \mathbf{A}_i$ . Note that the graph of this homomorphism  $G_{h_i} = \{(a, h_i(a)) : a \in \mathbf{S}_i\} \leq \mathbf{S}_i \times \mathbf{A}_i$  is an invariant relation. We define  $\Gamma'$  to be the union of  $\Gamma$  and all binary relations  $G_h$ . It is not hard to see that  $R' = \{(a_1, \dots, a_n) \in \mathbf{S}_1 \times \dots \times \mathbf{S}_n \mid (h_1(a_1), \dots, h_n(a_n)) \in R\}$  is invariant under  $\mathbf{A}$ . By assumption,  $R'$  has a pp-definition  $\phi'(x_1, \dots, x_n)$  of length in  $O(f(n))$ . The relation  $R$  can then be defined by the pp-formula  $\exists y_1, \dots, y_n (\bigwedge_{i=1}^n G_{h_i}(y_i, x_i) \wedge \phi'(y_1, \dots, y_n))$ , whose length is also in  $O(f(n))$ .

For the converse, let us consider relations  $R \leq \mathbf{A}_1 \times \dots \times \mathbf{A}_n$  with  $\mathbf{A}_i \in \text{HS}(\mathbf{A})$  from any relational basis  $\Gamma$  of  $\text{Inv}(\text{HS}(\mathbf{A}))$ . Then  $\mathbf{A}_i = h_i(\mathbf{S}_i)$ , for a subalgebra  $\mathbf{S}_i \leq \mathbf{A}$  and a homomorphism  $h_i: \mathbf{S}_i \rightarrow \mathbf{A}_i$ . As above,  $R' = \{(a_1, \dots, a_n) \in A^n \mid (h_1(a_1), \dots, h_n(a_n)) \in R\}$  is invariant under  $\mathbf{A}$ . It is not hard to see that  $\Gamma' = \{R' \mid R \in \Gamma\}$  is a relational basis of  $\text{Inv}(\mathbf{A})$ , and for any pp-definition  $\phi$  of a relation  $Q \leq \mathbf{A}^n$  over  $\Gamma$ , the formula  $\phi'$  obtained by replacing every occurrence of the symbol  $R$  by  $R'$  defines  $Q$  over  $\Gamma'$ . ◀



# The Online Simple Knapsack Problem with Reservation and Removability

Elisabet Burjons  

York University, Toronto, Canada

Matthias Gehnen<sup>1</sup>  

RWTH Aachen University, Germany

Henri Lotze  

RWTH Aachen University, Germany

Daniel Mock  

RWTH Aachen University, Germany

Peter Rossmanith  

RWTH Aachen University, Germany

---

## Abstract

In the *online simple knapsack problem*, a knapsack of unit size 1 is given and an algorithm is tasked to fill it using a set of items that are revealed one after another. Each item must be accepted or rejected at the time they are presented, and these decisions are irrevocable. No prior knowledge about the set and sequence of items is given. The goal is then to maximize the sum of the sizes of all packed items compared to an optimal packing of all items of the sequence.

In this paper, we combine two existing variants of the problem that each extend the range of possible actions for a newly presented item by a new option. The first is *removability*, in which an item that was previously packed into the knapsack may be finally discarded at any point. The second is *reservations*, which allows the algorithm to delay the decision on accepting or rejecting a new item indefinitely for a proportional fee relative to the size of the given item.

If both removability and reservations are permitted, we show that the competitive ratio of the *online simple knapsack problem* rises depending on the relative reservation costs. As soon as any nonzero fee has to be paid for a reservation, no online algorithm can be better than 1.5-competitive. With rising reservation costs, this competitive ratio increases up to the golden ratio ( $\phi \approx 1.618$ ) that is reached for relative reservation costs of  $1 - \frac{\sqrt{5}}{3} \approx 0.254$ . We provide a matching upper and lower bound for relative reservation costs up to this value. From this point onward, the tight bound by Iwama and Taketomi for the *removable knapsack problem* is the best possible competitive ratio, not using any reservations.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** online algorithm, knapsack, competitive ratio, reservation, preemption

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.29

## 1 Introduction

Online problems model situations where an algorithm receives an input piece-wise. The analysis of online problems does not revolve around how much time and space is used when solving the problem, but how *good* the solution provided by an online algorithm is with respect to an optimal offline counterpart. In classical online models, an online algorithm receives a piece of input or *request* and must output some irrevocable part of the solution before the next request arrives, without room for changing the output later on.

---

<sup>1</sup> corresponding author



The *simple knapsack problem* is, as the name suggests, a simplification of the classical *knapsack problem*: Given a set of items  $I$ , a size function  $w : I \rightarrow \mathbf{R}$  and a gain function  $g : I \rightarrow \mathbf{R}$ , the task is to find a subset of items  $S \subseteq I$  such that  $\sum_{i \in S} w(i) \leq 1$  and  $\sum_{i \in S} g(i)$  is maximized. In the *simple knapsack problem*,  $w : I \rightarrow [0, 1]$  and  $w(i) = g(i)$  for all items  $i \in I$ . The offline knapsack problem is a classical hard optimization problem and has been shown to be NP-complete [14]. Both the knapsack problem and the simple knapsack problem admit fully polynomial time approximation schemes [11].

In this work we study the *online simple knapsack problem*. Given a knapsack with capacity 1, an algorithm receives a request sequence  $S = \{x_1, \dots, x_n\}$  consisting of items of size smaller than or equal to 1, and must decide at each step whether to take such an item or reject it. The length of the sequence is initially unknown to the algorithm and the decision on each item must be made before the next item arrives. The decisions are irrevocable, and once an item has been packed or rejected, the decision cannot be reversed in later steps.

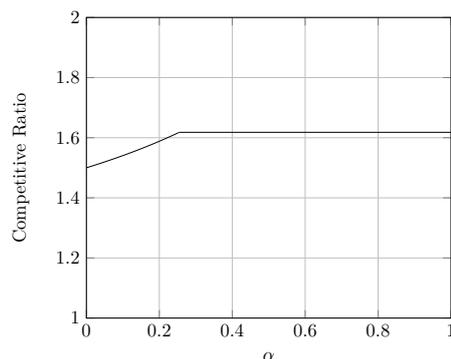
Competitive analysis was introduced by Sleator and Tarjan [19] as a way to analyze the performance of online algorithms. The *strict competitive ratio* of an online algorithm for an online maximization problem, such as the *online simple knapsack problem*, is the ratio between the gain obtained by an optimal offline algorithm and the gain obtained by the online algorithm in the worst-case instance. For an introduction to online analysis, we refer the reader to the standard book by Borodin and El-Yaniv [4].

Several variants of online models have been introduced to discern which problems are truly complex, or have a high information content, and which problems are simple in general but contain some rare worst-case instances yielding large competitive ratios. Some of these approaches, such as randomized algorithms [1], advice complexity [7, 10, 15], randomization of the adversary [6, 16, 18], etc. have been applied to various online problems, including the *online simple knapsack problem*, with some success.

In the general online setting, it is well-known [17] that there cannot be an algorithm for the online simple knapsack problem – and thus also for the online knapsack problem – with a bounded competitive ratio. Thus, most research so far has been focused on analyzing some variants of the problem or of the online model. Iwama and Taketomi [12] introduced a variant that adds the possibility to remove previously packed items from the knapsack, without any additional costs. This can be done at any point during the run of an algorithm. They showed a tight competitive ratio of the golden ratio  $\phi \approx 1.618$  for this model.

Recently Böckenhauer et al. [2] introduced a model for the online simple knapsack that adds the possibility to reserve items for a fee proportional to their size. In this model, if an item is reserved, the decision whether the item should be added to the knapsack can be postponed to a later point. However, if an algorithm decides to reserve an item, it has to pay some reservation cost  $c = \alpha x$  that depends on a given parameter  $\alpha$  between 0 and 1, and the item size  $x$ . Surprisingly, in this case, even for arbitrarily small values of the reservation parameter  $\alpha$ , the competitive ratio is not better than 2.

In this paper, we consider a variant of the online simple knapsack problem where an algorithm is allowed to reserve items (reservation) or alternatively to remove items previously placed into the knapsack (removability). Both of these alternatives allow an algorithm to postpone or alter decisions on whether to pack an object. Our reason to choose this variant is that it is a variant of the online simple knapsack with bounded competitive ratio below 2, the best ratio that the reservation model allowed. Some of the questions we wanted to answer is whether the reservations would help at all, and if so, if it is possible to achieve a competitive ratio approaching 1 for low reservation costs. We also want to know if the model behaves similarly to the secretary problem with reservation [5], where for large reservation costs the competitive ratio is the same as without reservations.



■ **Figure 1** A schematic plot of the competitive ratio with respect to the reservation costs  $\alpha$  for the online simple knapsack problem with reservation and removability.

Other variants of the knapsack problem include the work by Böckenhauer et al. [3], who studied the online simple knapsack problem with advice and randomization, and concluded that with one single advice or random bit the problem becomes 2-competitive, whereas further additional random bits do not improve the competitive ratio. Iwama and Zhang [13] looked at the problem with resource augmentation, which means that the size of the knapsack for the online problem is slightly larger than the knapsack size of the offline algorithm. Han, Kawase and Makino [8] considered a version where the items may be removed from the knapsack at the cost of a factor of the item size, and showed that if the factor is smaller than  $1/2$ , the competitive ratio is 2, and otherwise it is a function depending on the factor itself. This is different from the variant we consider, as in this case one pays to remove items from the knapsack, whereas in our version the algorithm pays to reserve items, but not to remove them. Another variant of the problem allows for a buffer of constant size, in which items may be intermediately stored. Han et al. [9] studied the case where the buffer has at least as much capacity as the knapsack itself; the items presented may be allocated into the buffer or irrevocably rejected and only in the last step are the items selected for placement into the knapsack. The reservation model has also been recently applied to the secretary problem [5], where one can achieve a competitive ratio as close to 1 as possible with diminishing values of the reservation cost. For large reservation costs, the competitive ratio is the same as in the model without reservation.

For the online simple knapsack problem with both removability and reservation, we provide tight upper and lower bounds of  $\frac{3-1.5\alpha}{2-1.5\alpha}$  for  $0 < \alpha \leq 1 - \sqrt{5}/3 \approx 0.2546$ , which goes from 1.5 to  $\phi$ . For larger values of  $\alpha$ , the competitive ratio stays constant at  $\phi$ . These bounds are depicted in Figure 1. This means that, even if the reservation costs are arbitrarily small the competitive ratio is worse than 1.5, and for any  $\alpha > 0.2546$  the best achievable competitive ratio is the one that can be achieved without reserving any items. This is in contrast to the behavior of the competitive ratio for the reservation model without removability, where the competitive ratio is constant at value 2 on the small range, and grows until being unbounded for large values of  $\alpha$ .

## 1.1 Problem Definition

We now formally define the online simple knapsack problem with reservation and removability.

Just as in the standard online simple knapsack problem, defined in the introduction, an algorithm solving the problem with reservation and removability is given a knapsack with capacity 1, and receives a request sequence  $S = \{x_1, \dots, x_n\}$  of items. Every item  $x_i$  of the

## 29:4 The Online Simple Knapsack Problem with Reservation and Removability

request sequence only has one parameter, the size, thus it is possible to refer to the size of each item as  $x_i$  while keeping the context clear. At the beginning of the request sequence, the algorithm is also given a parameter  $0 \leq \alpha \leq 1$  for the reservation costs. After receiving an item of size  $x$  the algorithm has three options. It can either decide to pack this item – assuming the total size of already packed items together with the new item is smaller than 1 –, reserve this item at cost  $\alpha x$ , or reject this item. Moreover, the algorithm can also decide to remove and reject any item currently in the knapsack at any point. The decision to reject an item is irrevocable.

Once the whole request sequence  $S$  has been processed, an algorithm  $A$  can decide to pack any reserved items, removing any items in the knapsack accordingly if necessary, as long as they fit into the knapsack. The goal of an algorithm is to maximize its total size of items packed in the knapsack after the whole instance has been processed minus the reservation costs, that is,

$$\text{gain}_S(A, \alpha) = \sum_{x_i \text{ packed}} x_i - \sum_{x_i \text{ reserved}} \alpha x_i .$$

The performance of an algorithm is measured against the optimal solution in a worst-case manner. Thus, the competitive ratio of an algorithm  $A$  is the highest ratio between the size of an optimal solution  $OPT$  and the gain of  $A$ , over all instances

$$c(A, \alpha) = \sup_S \left\{ \frac{\text{gain}_S(OPT, \alpha)}{\text{gain}_S(A, \alpha)} \right\} .$$

We now present matching upper and lower bounds for the competitive ratio of any algorithm solving the online simple knapsack problem with reservation and removability.

### 2 Lower Bound

First note that no algorithm can be better than optimal, in particular we have a lower bound of 1 for  $\alpha = 0$ .

In this section we show that no algorithm can reach a competitive ratio smaller than  $\min\{\frac{3-1.5\alpha}{2-1.5\alpha}, \phi\}$  for all  $0 < \alpha < 1$ . The ratio  $\frac{3-1.5\alpha}{2-1.5\alpha}$  is equal to  $\phi$  at  $\alpha = 1 - \frac{\sqrt{5}}{3} \approx 0.2546$ . For any larger values of  $\alpha$ , we prove that it is not possible to construct an algorithm that performs better than the strategy of Iwama and Taketomi for the knapsack with removability and without reservation [12].

► **Theorem 1.** *Given a parameter  $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$ , there exists no algorithm for online simple knapsack with reservation and removability achieving a competitive ratio better than  $\frac{3-1.5\alpha}{2-1.5\alpha}$ .*

**Proof.** We present an adversarial strategy to show that no algorithm can reach a competitive ratio of  $\frac{3-1.5\alpha}{2-1.5\alpha} - \varepsilon$  for any given  $\varepsilon > 0$ . For the following analysis, we will abbreviate  $\frac{2-1.5\alpha}{3-1.5\alpha}$  with  $k$ . Consider the following set of adversarial instances with  $0 < \delta \ll \varepsilon$ :

The adversary starts by presenting the first item  $x_1$  of size  $1 - k + \delta$ . In this situation an arbitrary algorithm  $A$  can choose between the following options:

**Case 1, Pack  $x_1$ .**  $A$  packs the first item of size  $1 - k + \delta$  into the knapsack. The adversary next presents the second item  $x_2$  of size  $k + \delta$  that barely does not fit together with the first item.

**Case 1.1, Remove  $x_1$  and pack  $x_2$ .**  $A$  discards the item of size  $1 - k + \delta$  and packs  $x_2$ .

The adversary then presents an item of size  $1 - x_1 = k - \delta$ . This would perfectly fit together with the discarded item  $x_1$ , so  $OPT$  has a solution of size 1. Since this item does not fit together with  $x_2$ ,  $A$  only holds  $x_2$  as the larger one of the two items, which results in the desired competitive ratio.

**Case 1.2, Reserve  $x_2$ .**  $A$  reserves the item of size  $k + \delta$ . The adversary presents the item  $x_3$  of size  $k + \delta^2$ .

**Case 1.2.1, Remove  $x_1$  and pack  $x_3$ .** Assuming  $A$  discards the first item that is still held in its knapsack to pack the item of size  $k + \delta^2$ . Similar to Case 1.1, the adversary then presents the counterpart to the discarded item of size  $1 - x_1 = k - \delta$ .  $A$  cannot pack its reserved item together with the newly packed item, since  $k - \delta + k + \delta^2 > 1$  for all  $\alpha \in (0, 1 - \frac{\sqrt{5}}{3})$ . The gain of  $A$  is even smaller than in Case 1.1, due to the costs of reserving an item.

**Case 1.2.2, Reserve  $x_3$ .**  $A$  reserves the item of size  $k + \delta^2$ . The adversary presents  $x_4$  of size  $k + \delta^3$  and will continue to present items  $x_{j+1}$  of size  $k + \delta^j$  as long as  $A$  reserves those items. At some point  $A$  must stop reserving these items due to the accumulating reservation costs exceeding any possible gain. As soon as  $A$  rejects an item, or discards the item in its knapsack to pack an item, the case can be handled analogously to 1.2.1 or 1.2.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

**Case 1.2.3, Reject  $x_3$ .** The algorithm does not pack the item of size  $k + \delta^2$ . The adversary presents an item of size  $1 - x_3$ . So while  $OPT$  reaches 1,  $A$  can at the most pack  $x_1$  and the item of size  $1 - x_3$ . Together, with subtracted reservation costs, the algorithm then has a gain of  $2 - \frac{4-3\alpha}{3-1.5\alpha} - \alpha k$ . A simple analysis shows that this yields a ratio higher than  $\frac{3-1.5\alpha}{2-1.5\alpha}$ .

**Case 1.3, Reject  $x_2$ .**  $A$  does not pack or reserve the item of size  $k + \delta$ . No further items are presented. The algorithm thus only holds  $x_1$ , while an optimal solution would be to pack the item  $x_2$ . The ratio of  $k$  to  $1 - k$  is worse than  $\phi$  for every  $\alpha < 1 - \frac{\sqrt{5}}{3}$  and therefore, in particular, larger than the desired ratio.

**Case 2, Reserve  $x_1$ .**  $A$  reserves the first item of size  $1 - k + \delta$ . The adversary presents the item  $x'_2$  of size  $\frac{2}{3} + 2\delta$ .

**Case 2.1 Pack  $x'_2$ .** The algorithm packs the item of size  $\frac{2}{3} + 2\delta$ . The next item is  $x'_3$  of size  $\frac{1}{3} + 2\delta$ .

**Case 2.1.1 Remove  $x'_2$  and pack  $x'_3$ .** The algorithm discards the item of size  $\frac{2}{3} + 2\delta$  from its knapsack and instead packs the item of size  $\frac{1}{3} + 2\delta$ . The adversary next presents an item of size  $1 - x'_2$ .  $OPT$  then has a solution of size 1.  $A$  can at most reach a gain of  $x_1 + x'_3 - \alpha x_1 = \frac{1}{3} + 1 - k + 3\delta - \alpha(1 - k) = k$ .

**Case 2.1.2 Reserve  $x'_3$ .**  $A$  reserves the item of size  $\frac{1}{3} + 2\delta$ . The adversary continues to present items  $x'_{j+1}$  of size  $\frac{1}{3} + \delta + \delta^{j-1}$ , as long as  $A$  reserves those items. As soon as  $A$  rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.1.1 or 2.1.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

**Case 2.1.3 Reject  $x'_3$ .** The algorithm does not pack the item of size  $\frac{1}{3} + 2\delta$ . The adversary then presents an item of size  $1 - x'_3$ . Since  $x'_3$  was the smallest item of all that have been presented so far, the counterpart will not fit together with any other item of  $A$ . Therefore  $OPT$  has a gain of 1, whereas the largest packing of  $A$  consists of the item  $x'_2$  which is of smaller size than  $k$ .

**Case 2.2 Reserve  $x'_2$ .**  $A$  reserves the item of size  $\frac{2}{3} + 2\delta$ . The adversary presents  $x''_2$  of size  $\frac{2}{3} + \delta + \delta^2$ .

**Case 2.2.1 Pack  $x''_2$ .**  $A$  packs the item of size  $\frac{2}{3} + \delta + \delta^2$  into the knapsack. The next item is  $x'_3$  of size  $\frac{1}{3} + 2\delta$ .

**Case 2.2.1.1 Remove  $x''_2$  and pack  $x'_3$ .** The algorithm discards the item of size  $\frac{2}{3} + \delta + \delta^2$  from its knapsack and instead packs the item of size  $\frac{1}{3} + 2\delta$ . The adversary next presents an item of size  $1 - x''_2$ .  $OPT$  then has a solution of size 1.  $A$  can at most reach a gain of  $x_1 + x'_3 - \alpha x_1 = \frac{1}{3} + 1 - k + 3\delta - \alpha(1 - k) = k$ .

**Case 2.2.1.2 Reserve  $x'_3$ .**  $A$  reserves the item of size  $\frac{1}{3} + 2\delta$ . The adversary continues to present items  $x'_{j+1}$  of the size  $\frac{1}{3} + \delta + \delta^{j-1}$ , as long as  $A$  reserves those. As soon as  $A$  rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.2.1.1 or 2.2.1.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

**Case 2.2.1.3 Reject  $x'_3$ .** The algorithm does not pack the item of size  $\frac{1}{3} + 2\delta$ . The adversary then presents an item of size  $1 - x'_3$ . Since  $x'_3$  was the smallest item of all that have been presented so far, the counterpart will not fit together with any other item of  $A$ . Therefore  $OPT$  has a gain of 1, whereas the largest packing of  $A$  consists of the item  $x'_2 < k$ .

**Case 2.2.2 Reserve  $x''_2$ .** The algorithm reserves the item of size  $\frac{2}{3} + \delta + \delta^2$ . The adversary continues to present items  $x''_j$  of the size  $\frac{2}{3} + \delta + \delta^j$ , as long as  $A$  reserves these items. As soon as  $A$  rejects an item or discards the item already in its knapsack to pack an item, these cases can be handled analogously to 2.2.1 or 2.2.3. The gain of the algorithm will be lower than in these cases due to the added reservation costs.

**Case 2.2.3 Reject  $x''_2$ .**  $A$  does not pack the item of size  $\frac{2}{3} + \delta + \delta^2$ . The adversary then presents an item of size  $1 - x''_2$ . The gain of  $OPT$  is then 1. The maximum gain of  $A$  is then  $x_1 + 1 - x''_2 - \alpha(x_1 + x'_2) = 1 - k + \frac{1}{3} - \delta - \delta^2 - \alpha(1 - k + \frac{1}{3} - \delta - \delta^2)$  which is slightly smaller than the gain of  $A$  in Case 2.1.1 due to increased reservation costs and thus especially smaller than  $k$ .

**Case 2.3 Reject  $x'_2$ .** The algorithm does not pack the item of size  $\frac{2}{3} + 2\delta$ . No further items are presented. Similarly to Case 1.3, the algorithm only holds  $x_1$ , while an optimal packing would be  $x'_2$ . Since even  $x'_2 > x_2$ , the same argumentation from Case 1.3 concludes this case.

**Case 3, Reject  $x_1$ .** No further items are presented. Therefore the optimal solution is exactly the item  $x_1 = 1 - k$ , while  $A$  has packed no items. The competitive ratio is then unbounded.  $\blacktriangleleft$

It is not surprising that for increasing reservation costs, the competitive ratio is increasing, too. In particular, it is easy to see that the competitive ratio cannot decrease if the reservation costs increase.

**► Corollary 2.** For  $\alpha > 1 - \frac{\sqrt{5}}{3}$ , no algorithm can be better than  $\phi$ -competitive for the online simple knapsack problem with reservation and removability.

**Proof.** The competitive ratio of Theorem 1 reaches  $\phi$  at  $\alpha = 1 - \frac{\sqrt{5}}{3}$ . For every  $\alpha > 1 - \frac{\sqrt{5}}{3}$  it is possible to present the lower bound with exactly the same items as in Theorem 1 for  $\alpha = 1 - \frac{\sqrt{5}}{3}$ . The only difference are the higher costs for every reservation. Thus, in every case, the gain of an optimal offline solution stays the same, while the algorithm gets at most the gain of the case  $\alpha = 1 - \frac{\sqrt{5}}{3}$ .  $\blacktriangleleft$

### 3 Upper Bound

If  $\alpha = 0$ , then the problem becomes an offline problem, since an algorithm can reserve every item for free. Thus a competitive ratio of 1 is achievable. Trivially, this matches the lower bound.

However, as we will see now, as soon as the value of  $\alpha$  is positive, the best achievable competitive ratio is worse than 1.5-competitive.

To handle the range of  $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$ , we present Algorithm 1 with a competitive ratio of  $\frac{3-1.5\alpha}{2-1.5\alpha}$ , matching the lower bound established in the previous subsection.

Algorithm 1, analyzed in this section, can be split in three phases. All of the phases work similarly: First, if small items arrive, they are packed since they will not cause harm (we can throw them out at any time).

Then there are two kind of interesting ranges for medium-sized items: A large-medium range, where the algorithm tries to keep the smallest items to maintain the possibility to get fitting counterparts (similar to [12]). And a small-medium range, where an item gets reserved, because otherwise the ratio between large and small items in the large-medium range, where only the smallest items are kept, gets too large by itself. After the first reservation, the algorithm continues in the second phase; after the second reservation in the third phase.

Finally, very large items, which are sufficiently large to guarantee the desired competitive ratio by themselves, are the easiest case since the algorithm can just pack these items to achieve the desired competitive ratio immediately.

The main difference between the three phases is, that the ranges for the different actions are shifted – due to the ability to use the reserved items, but also due to the paid reservation costs that need to be compensated. Due to these shifts, in the third phase there is no small-medium range, and there is no need to reserve any more items, thus Algorithm 1 reserves at most two items.

► **Theorem 3.** *For  $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$ , there exists an algorithm solving the online simple knapsack problem with reservation and removability that is not worse than  $\frac{3-1.5\alpha}{2-1.5\alpha}$  competitive.*

**Proof.** We prove that Algorithm 1 achieves the competitive ratio on the statement. For ease of notation, we define  $c := \frac{3-1.5\alpha}{2-1.5\alpha}$ . If Algorithm 1 is at any point able to pack a knapsack such that, even when paying the reservation costs, the desired competitive ratio is reached, it stops afterwards. This is the case in lines 4, 9 and 16.

Depending on the instance, Algorithm 1 reserves zero, one or two items. In the following, we show that the algorithm yields the desired competitive ratio in those three cases.

**Case 1: Algorithm 1 ends in line 4 or 22, before the first reservation is made.** When presented with a new item, the algorithm first checks if this item together with a subset of its packed items already yields a packing of size  $\frac{1}{c}$ . In the following analysis, assume that this is not the case, since if this were the case, the algorithm would have achieved the desired competitive ratio anyways.

Items that are smaller than  $1 - \frac{1}{c}$  are packed by the algorithm in line 5. Items with sizes between  $1 - \frac{1}{c}$  and  $\frac{1}{c^2}$  will be reserved in line 7, if the condition in line 4 does not hold: We investigate this case as Case 2.

The first item of size between  $\frac{1}{c^2}$  and  $\frac{1}{c}$  is packed. If further items in this range arrive, only the smallest item of this range is packed (or kept) in the knapsack and the other items from this range are discarded. If an item  $x_k$  in this range, which should be kept in the knapsack according to line 6, does not fit, it must be due to the items already in the knapsack, which are each smaller than  $1 - \frac{1}{c}$ . Thus, in this case, it is possible to combine this item  $x_k$  with a subset of the small items in the knapsack to get a packing of size at least  $\frac{1}{c}$ , triggering the end of the packing in line 4.

■ **Algorithm 1** Upper bound Algorithm for  $0 < \alpha \leq 1 - \frac{\sqrt{5}}{3}$ .  $\text{OPT}(S)$  denotes the size of the optimal packing of the set  $S$ ,  $c := \frac{3-1.5\alpha}{2-1.5\alpha}$ .

---

```

1:  $K := \emptyset; R := \emptyset$ 
2: for  $k = 1, \dots, n$  do
3:   if  $|R| = 0$  then
4:     if  $\text{OPT}(K \cup \{x_k\}) \geq \frac{1}{c}$  then Pack  $\text{OPT}(K \cup \{x_k\})$  END
5:     else if  $x_k \leq 1 - \frac{1}{c}$  then  $K := K \cup \{x_k\}$ ;
6:     else if  $x_k \geq \frac{1}{c^2}$  then Keep only smallest  $x'_k \in K \cup x_k$  with  $x'_k \in [\frac{1}{c^2}, \frac{1}{c}]$ ;
7:     else  $(1 - \frac{1}{c} < x_k < \frac{1}{c^2})$   $R := R \cup x_k, x_f := x_k$ ;
8:   else if  $|R| = 1$  then
9:     if  $\text{OPT}(K \cup \{x_f, x_k\}) \geq \frac{1}{c} + \alpha x_f$  then Pack  $\text{OPT}(K \cup \{x_f, x_k\})$  END
10:    else if  $1 - \frac{1}{c} - \alpha x_f < x_k \leq \frac{1}{3}$  then  $R := R \cup x_k, g := k$ ;
11:    else if  $x_k \leq 1 - \frac{1}{c} - \alpha x_f$  then  $K := K \cup \{x_k\}$ ;
12:    else Keep only smallest  $x'_k \in K \cup x_k$  with  $x'_k \in [1 - x_f, \frac{1}{c} + \alpha x_f]$ ;
13:   else
14:      $M := \emptyset; L := \emptyset$ 
15:     if  $\text{OPT}(K \cup \{x_f, x_g, x_k\}) \geq \frac{1}{c} + \alpha(x_f + x_g)$  then
16:       Pack  $\text{OPT}(K \cup \{x_f, x_g, x_k\})$  END
17:     else if  $x_k \leq 1 - \frac{1}{c} + \alpha(x_f + x_g)$  then  $K := K \cup \{x_k\}$ ;
18:     else if  $1 - x_g - x_f < x_k < \frac{1}{c} + \alpha(x_g + x_f) - x_f$  then
19:       Keep  $x_k$  if  $x_k + x_g < \min(x_l \in L) \wedge x_k < \min(x_m \in M)$ ;  $M := M \cup x_k$ ;
20:     else if  $1 - x_g < x_k < \frac{1}{c} + \alpha(x_f + x_g)$  then
21:       Keep  $x_k$  if  $x_k + x_g < \min(x_m \in M) \wedge x_k < \min(x_l \in L)$ ;  $L := L \cup x_k$ ;
22:   Pack  $\text{OPT}(K \cup R)$ 
    
```

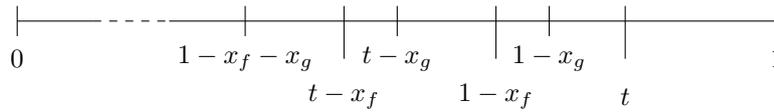
---

On the other hand, if the instance ends before reserving the first item, the only possible difference between an optimal solution and a solution of the algorithm is that the algorithm only has the smallest item in the range between  $\frac{1}{c^2}$  and  $\frac{1}{c}$ , while the optimal solution might have a different item in this range. Since the smallest item in this range is at least of size  $\frac{1}{c^2}$  and the largest item is at most of size  $\frac{1}{c}$ , the ratio is smaller than  $\frac{1/c}{1/c^2} = c$ . Additional small items can be part of both the optimal solution and the solution provided by the algorithm, which only makes the ratio smaller.

Note that an optimal solution cannot contain two items in the range between  $\frac{1}{c^2}$  and  $\frac{1}{c}$ , since the algorithm would also be able to combine the two items in this range. The algorithm always keeps the smallest possible item within the range, a second item within the range that could fit into the knapsack would trigger the condition in line 4, since two items of size at least  $\frac{1}{c^2}$  are sufficient to reach  $\frac{1}{c}$  ( $\frac{2}{c^2} \geq \frac{1}{c}$  for  $c \leq 2$ ).

**Case 2: Algorithm ends in line 9 or 22 with one reserved item.** Assume one item  $x_f$  in the range  $1 - \frac{1}{c}$  to  $\frac{1}{c^2}$  was reserved. The reservation costs for this item are  $\alpha x_f$ . Thus, the algorithm needs to pack at least  $\frac{1}{c} + \alpha x_f$  to guarantee the desired competitive ratio independently to the gain achieved by an optimal solution.

For items smaller than  $1 - \frac{1}{c} - \alpha x_f$  the same argument for the small items in Case 1 holds: These items can be packed in a greedy manner until the threshold  $\frac{1}{c} + \alpha x_f$  is reached. An item of size between  $1 - \frac{1}{c} - \alpha x_f$  and  $\frac{1}{3}$  will be reserved in line 10, if the algorithm cannot pack a knapsack of size  $\frac{1}{c} + \alpha x_f$  at this point. This case is discussed in the next part of the proof, as Case 3.



■ **Figure 2** A schematic plot of the size ranges in the algorithm when two items have already been reserved. The targeted size for the items is  $t = \frac{1}{c} + \alpha(x_g + x_f)$ .

Note that it is sufficient to combine the first reserved item with a single item of size at least  $\frac{1}{3}$  to reach the desired competitive ratio, even after paying the reservation costs, as

$$\frac{1}{3} + (1 - \alpha)x_f \geq \frac{1}{3} + (1 - \alpha)\left(1 - \frac{1}{c}\right) = \frac{1}{c},$$

in the considered range of  $\alpha$ . In addition, any item that reaches  $\frac{1}{c} + \alpha x_f$  either by its own or together with small items is sufficient as well. Thus, with every item between  $\frac{1}{3}$  and  $1 - x_f$ , or larger than  $\frac{1}{c} + \alpha x_f$ , the threshold is reached already and the algorithm stops in line 9.

We are left to consider items of sizes between  $1 - x_f$  and  $\frac{1}{c} + \alpha x_f$ , which will reach line 12 in the algorithm. With the items in this range, we follow the same strategy as for the large-medium items before the first reservation: The smallest of those items will be kept in the knapsack. This works for the same reasons as in Case 1, as explained below.

Assume the instance stops here and the algorithm was not able to reach  $\frac{1}{c} + \alpha x_f$ : The only difference between the solution of the algorithm and an optimal solution is that the algorithm sticks to the smallest item in the range between  $1 - x_f$  and  $\frac{1}{c} + \alpha x_f$ , which gives us the worst possible case. Note that no item of size smaller than  $1 - x_f$  can be rejected before the first reservation, since it otherwise would have been packed in combination with  $f$ . The ratio between  $\frac{1}{c} + \alpha x_f$  and  $1 - x_f$  is less than  $c$ , since  $x_f$  is at most  $\frac{1}{c^2}$ . Again, additional small items can be added to both the optimal solution and the solution provided by the algorithm, only decreasing the ratio.

**Case 3: Algorithm ends in line 16 or 22 with two reservations made.** Assume two items were reserved, the first item  $x_f$  of size between  $1 - \frac{1}{c}$  and  $\frac{1}{c^2}$ , and the second item  $x_g$  of size between  $1 - \frac{1}{c} - \alpha x_f$  and  $\frac{1}{3}$ . Therefore, the algorithm has already paid  $\alpha(x_g + x_f)$  as reservation costs, and now needs a packing of size  $\frac{1}{c} + \alpha(x_g + x_f)$  to guarantee the competitive ratio of  $c$  independent of any optimal solution size.

First note that, an item or a subset of items is sufficient to reach  $\frac{1}{c} + \alpha(x_f + x_g)$  in three cases: it either reaches the threshold by its own, reaches the threshold together with one of the reserved items, or with the two reserved items combined.

To reach the threshold with one of the reserved items, a subset of items must have a size between  $\frac{1}{c} + \alpha(x_f + x_g) - x_f$  and  $1 - x_f$ , or between  $\frac{1}{c} + \alpha(x_f + x_g) - x_g$  and  $1 - x_g$  (depending on which of the reserved items will be used to combine this subset with). The subsets sizes, that combined with  $x_f$  are sufficient to reach the necessary threshold can obviously be smaller than the subsets that can be combined with  $x_g$ , since  $x_g < \frac{1}{3}$  and  $x_f > \frac{1}{3}$ . With simple analysis, it can be shown that for any  $\alpha$  between 0 and  $1 - \frac{\sqrt{5}}{3}$  and for every choice of  $x_g$  and  $x_f$  in the given ranges,  $1 - x_g$  must be larger than  $\frac{1}{c} + \alpha(x_f + x_g) - x_f$ . Therefore these intervals overlap, as displayed in Figure 2.

It follows that there are three intervals in which an item can be presented and the algorithm might not stop immediately:

1. Items smaller than  $\frac{1}{c} + \alpha(x_g + x_f) - x_g - x_f$  are too small to reach the necessary size in combination with both of the reserved items. Note that those are smaller than  $1 - \frac{1}{c} - \alpha(x_g + x_f)$ , thus they can be packed greedily. As soon as the knapsack is overfull due to those items a subset of the small items can be selected to reach  $\frac{1}{c} + \alpha(x_g + x_f)$ . This argumentation is analogous to the small items argument in the two previous cases.

## 29:10 The Online Simple Knapsack Problem with Reservation and Removability

2. Items between  $1 - x_g - x_f$  and  $\frac{1}{c} + \alpha(x_g + x_f) - x_f$ , which are too large to fit together with both reserved items, but too small to reach the threshold in combination with the largest reserved item. We will call those items *mid-sized*.
3. Items that cannot be combined any of the reserved items but also do not reach the threshold by themselves, i.e., items between  $1 - x_g > \frac{2}{3}$  and  $\frac{1}{c} + \alpha(x_f + x_g)$ . Those items will be called *large*.

As we have already justified, the small items in 1. are not a concern. We argue now that the desired competitive ratio is also achieved when mid-sized and large items appear in the request sequence after reserving two items.

The algorithm keeps either the smallest large item, or the smallest mid-sized item. This is dependent on the situation, as detailed in lines 18 to 21: If the smallest large item is smaller than the smallest mid-sized item in combination with  $x_g$ , the algorithm keeps the smallest large item. On the other hand, if the smallest mid-sized item in combination with  $x_g$  is smaller than the smallest large item, the algorithm keeps the smallest mid-sized item. Note that we can assume that every large item is larger than  $\frac{2}{3}$ , since otherwise it can be combined with  $x_g$ , reaching the threshold  $\frac{1}{c} + \alpha(x_g + x_f)$ .

Therefore the following crucial observation holds: If the algorithm is able to pack either one large item together with  $x_f$ ,  $x_g$  or a mid-sized item, or the algorithm is able to pack three items among  $x_f$ ,  $x_g$  and mid-sized items, the algorithm will also reach the desired threshold and competitive ratio.

To see this, first note that every mid-sized item of size smaller than or equal to  $\frac{1}{3}$  will be packed into the knapsack and will never be replaced by a large item. This is the case because every large item must be larger than  $\frac{2}{3}$ , and an item of size smaller than  $\frac{1}{3}$  in combination with  $x_g$  must be smaller or equal than  $\frac{2}{3}$ .

Assume that, at some point, an arbitrary algorithm is about to pack the last item of a combination of one large and one mid-sized or reserved item, or a combination of three mid-sized or reserved items. Due to the construction and the selection of items before, we are able to do so as well:

- If the last item is a large item, the algorithm is able to pack such a combination, since the smallest mid-sized item is in the knapsack if it is smaller than the reserved  $x_g$ . Otherwise the reserved  $x_g$  is available for a combination.
- If the last item is a mid-sized one, it fits together with the current combination of items selected by Algorithm 1 (either two mid-sized or one large item), since it is the smallest possible combination by construction.

A simple calculation then shows that every such combination of items (one large and one mid-sized/reserved item, or three mid-sized/reserved items) is of size at least  $\frac{1}{c} + \alpha(x_g + x_f)$ , even if chosen as small as possible.

The only thing left to prove is that even if there is no such combination, the algorithm still reaches the designated competitive ratio:

First note that large items are only rejected or removed from the knapsack in the first two phases. Those items had at most size  $\frac{1}{c} + \alpha x_f$ . Items that remained in the knapsack can be considered as either small items that can be packed in a greedy manner, or as large items. If there were an item that is not in the allowed range for large or small items in this phase, it would have been used to reach the threshold for a guaranteed competitive ratio immediately, just like it would have been possible if the item was presented in the third phase.

By using mid-sized and large items together with the reserved ones, every optimal solution can only reach  $\frac{1}{c} + \alpha(x_g + x_f)$ , while Algorithm 1 cannot be worse than  $x_g + x_f$  on any instance. An easy calculation shows that even for smallest possible sizes of  $x_g$  and  $x_f$ , their

combination is sufficient to reach the competitive ratio of  $c$ , since an optimal solution cannot be better than  $\frac{1}{c} + \alpha(x_g + x_f)$ . Since Algorithm 1 can use small items in the same manner as an optimal solution, the appearance of small items cannot worsen the ratio. ◀

This concludes the analysis for an upper bound that matches the lower bound for values of  $\alpha$  smaller than  $\alpha \geq 1 - \sqrt{5}/3$ .

For any  $\alpha \geq 1 - \sqrt{5}/3$ , note that the algorithm for knapsack with removable objects (without reservation) presented by Iwama and Taketomi [12] achieves a competitive ratio of  $\phi$ , thus matching our lower bound as well.

## 4 Concluding Remarks

The upper and lower bounds for the competitive ratio of the online simple knapsack with reservation and removability show that, as long as the value of  $\alpha$  is positive the competitive ratio is above 1.5. For large values of  $\alpha$  the competitive ratio stays the same as in the removability model without reservation, which is somehow intuitive, showing that at some point reserving items is not worth the costs. More surprising are the results for small values of  $\alpha$ . The algorithm with reservation that achieves the desired upper bound rarely makes use of the reservation, even in the case with very small  $\alpha$  only two items are reserved in total. This behavior is similar to that of online simple knapsack with reservations but without removability for small values of  $\alpha$ .

For future research, a natural variation of the problem would be to allow items to be reserved after they were added to the knapsack, i.e. to allow removing items from the knapsack and reserve them instead of only having the option to discard them (like we considered here).

It is known that the classic general online knapsack does not yield a finite competitive ratio and also the option to remove items of the knapsack does not help. Thus, it remains an interesting question if the reservation model changes the situation. We think that, at least for small reservation costs, it should be possible to achieve a constant competitive ratio, but this is still open and could also be a topic for further research.

---

## References

- 1 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in online algorithms (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 379–386. ACM, 1990. doi:10.1145/100216.100268.
- 2 Hans-Joachim Böckenhauer, Elisabet Burjons, Juraj Hromkovic, Henri Lotze, and Peter Rossmanith. Online simple knapsack with reservation costs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.16.
- 3 Hans-Joachim Böckenhauer, Dennis Komm, Richard Kráľovič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014. doi:10.1016/j.tcs.2014.01.027.
- 4 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- 5 Elisabet Burjons, Matthias Gehnen, Henri Lotze, Daniel Mock, and Peter Rossmanith. The secretary problem with reservation costs. In Chi-Yeh Chen, Wing-Kai Hon, Ling-Ju Hung, and Chia-Wei Lee, editors, *Computing and Combinatorics – 27th International Conference, COCOON 2021, Tainan, Taiwan, October 24-26, 2021, Proceedings*, volume 13025 of *Lecture Notes in Computer Science*, pages 553–564. Springer, 2021. doi:10.1007/978-3-030-89543-3\_46.

- 6 Elisabet Burjons, Juraj Hromkovic, Rastislav Královic, Richard Královic, Xavier Muñoz, and Walter Unger. Online graph coloring against a randomized adversary. *Int. J. Found. Comput. Sci.*, 29(4):551–569, 2018. doi:10.1142/S0129054118410058.
- 7 Stefan Dobrev, Rastislav Královic, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Theor. Informatics Appl.*, 43(3):585–613, 2009. doi:10.1051/ita/2009012.
- 8 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014. doi:10.1007/s00453-013-9822-z.
- 9 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 28:1–28:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ISAAC.2019.28.
- 10 Juraj Hromkovic, Rastislav Královic, and Richard Královic. Information complexity of online problems. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer, 2010. doi:10.1007/978-3-642-15155-2\_3.
- 11 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.
- 12 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 293–305. Springer, 2002. doi:10.1007/3-540-45465-9\_26.
- 13 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Inf. Process. Lett.*, 110(22):1016–1020, 2010. doi:10.1016/j.ipl.2010.08.013.
- 14 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2\_9.
- 15 Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi:10.1007/978-3-319-42749-2.
- 16 Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30(1):300–317, 2000. doi:10.1137/S0097539796299540.
- 17 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995. doi:10.1007/BF01585758.
- 18 Prabhakar Raghavan. A statistical adversary for on-line algorithms. In Lyle A. McGeoch and Daniel Dominic Sleator, editors, *On-Line Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, February 11-13, 1991*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–84. DIMACS/AMS, 1991. doi:10.1090/dimacs/007/05.
- 19 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update rules. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1984, Washington, DC, USA*, pages 488–492. ACM, 1984. doi:10.1145/800057.808718.

# Parikh One-Counter Automata

Michaël Cadilhac ✉ 

DePaul University, Chicago, IL, USA

Arka Ghosh ✉

University of Warsaw, Poland

Guillermo A. Pérez ✉ 

University of Antwerp – Flanders Make, Belgium

Ritam Raha ✉ 

University of Antwerp – Flanders Make, Belgium

LaBRI, University of Bordeaux, France

---

## Abstract

Counting abilities in finite automata are traditionally provided by two orthogonal extensions: adding a single counter that can be tested for zeroness at any point, or adding  $\mathbb{Z}$ -valued counters that are tested for equality only at the end of runs. In this paper, finite automata extended with both types of counters are introduced. They are called Parikh One-Counter Automata (POCA): the “Parikh” part referring to the evaluation of counters at the end of runs, and the “One-Counter” part to the single counter that can be tested during runs.

Their expressiveness, in the deterministic and nondeterministic variants, is investigated; it is shown in particular that there are deterministic POCA languages that cannot be expressed without nondeterminism in the original models. The natural decision problems are also studied; strikingly, most of them are no harder than in the original models. A parametric version of nonemptiness is also considered.

**2012 ACM Subject Classification** Theory of computation → Grammars and context-free languages

**Keywords and phrases** Parikh automata, Context-free languages, One-counter automata

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.30

**Funding** Belgian FWO “SAILor” project (G030020N) & NCN grant 2019/35/B/ST6/02322.

## 1 Introduction

Extensions of finite automata abound in the literature, with a traditional common goal: To find computational models with good expressiveness for which relevant problems are decidable. The impetus lies in formal verification: Can I express my process using that new model, then formally check that it does not have “bad” behaviors? Typically, processes are thus implemented with expressive models, while bad behaviors can be represented using a regular language. To answer the verification question, the key computational problem is then *inclusion in a regular language* (are all the executions of my process not bad?).

A common approach to extending finite automata is to equip them with counters or some sort of counting abilities. The literature crystallizes around two main extensions:

- Adding a *single* counter which can be tested for zeroness throughout the run. A typical language that such an extension can recognize is  $L_1 = \{a^n b^n \mid n \geq 0\}^*$  (mind the star!).
- Adding any constant number of counters, but they can only be tested for zeroness or equality a bounded number of times (during the run or at the end only, these variants being equivalent for nondeterministic machines). This includes reversal-bounded counter machines [14, 6] and Parikh automata [16, 4, 5], which, incidentally, are equally expressive.

A typical language in these extensions is  $L_2 = \{a^n b^n c^n \mid n \geq 0\}$ .



© Michaël Cadilhac, Arka Ghosh, Guillermo A. Pérez, and Ritam Raha;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 30; pp. 30:1–30:15

Leibniz International Proceedings in Informatics



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

Unsurprisingly,  $L_1$  cannot be recognized using the second extension, while  $L_2$  cannot be recognized with the first. A natural extension is, thus, to combine these two approaches to counting into a single model, prompting the questions: **1.** Is the expressiveness of the combined model more interesting than just the union of the two original models; and **2.** Are the good decidability properties of the original models retained in the combined one?

Here, we report on this extension. The model, called *Parikh One-Counter Automata* (POCA), consists in a finite automaton with one counter which can be tested for zeroness and any number of  $\mathbb{Z}$ -valued counters that are checked at the end of the run using a Presburger formula (an arithmetic formula of first-order logic with addition).<sup>1</sup> We contribute:

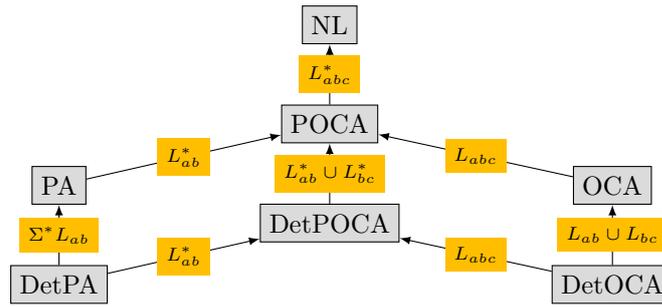
- A (de)pumping lemma allowing for the study of the limits of the expressiveness of POCA. Pumping in one-counter automata is fairly simple, as it follows standard arguments for pushdown automata. In Parikh automata (i.e., POCA without the unbounded counter), any cycle taken twice can be moved around without changing membership. However, combining these two properties for POCA proves to be a great technical challenge.
- A complete picture of the relationships between POCA, one-counter automata, and Parikh automata, in their nondeterministic and deterministic variants. In addition to separation of the classes of languages under consideration, we observe that some languages that are only *nondeterministic* for both Parikh automata and one-counter automata turn out to be *deterministic* for POCA. This is of special interest in the context of verification since many problems are undecidable for nondeterministic machines but decidable for deterministic ones. We also study how the base models of Parikh automata and one-counter automata are “embedded” in POCA (Theorem 15, the statement of which should be clear at this point of the Introduction).
- A study of the decision problems for POCA and its deterministic variant. Strikingly, emptiness and inclusion in a regular language are no more complex than with Parikh automata: coNP-complete. We also study *parametric* POCA, in which parameters  $x, y, \dots$  can be used in the counter updates (as  $+x, -y$ , for instance), and show that it is undecidable whether, for all parameter values, the language of the POCA is nonempty. We relate this problem to considerations on arithmetic theories since it is one of the main motivations behind the study of parametric models [1].

	$\cup$	$\cap$	$-$	$\cdot$	$h$	$h^{-1}$	$L \neq \emptyset$	$L = \Sigma^*$	$L_1 \subseteq L_2$	$L_1 = L_2$
DetPA	Y	Y	Y	N	N	Y	NP-c	coNP-c	coNP-c	coNP-c
DetOCA	N	N	Y	N	N	Y	NL-c	NL-c	Undec	NL-c
DetPOCA	N	N	Y	N	N	Y	NP-c	coNP-c	Undec	?
PA	Y	Y	N	Y	Y	Y	NP-c	Undec	Undec	Undec
OCA	Y	N	N	Y	Y	Y	NL-c	Undec	Undec	Undec
POCA	Y	N	N	Y	Y	Y	NP-c	Undec	Undec	Undec

Thm. 18 points to the  $\cup$  and  $\cap$  cells of the top row.  
 Thm. 20 points to the  $\cdot$  and  $h$  cells of the top row.  
 Thm. 19 points to the  $\cup$  and  $\cap$  cells of the bottom row.  
 Thm. 21 points to the  $\cdot$  and  $h$  cells of the bottom row.  
 Thm. 23 points to the  $L \neq \emptyset$  and  $L = \Sigma^*$  cells of the bottom row.  
 Cor. 22 points to the  $L_1 \subseteq L_2$  and  $L_1 = L_2$  cells of the bottom row.

■ **Figure 1** Closure properties and complexity results. Results about Parikh automata (PA) and one-counter automata (OCA) are from the literature [7, 16, 5, 17, 21, 2]. The left side of the table lists closure properties;  $h$  and  $h^{-1}$  mean closure under morphisms and inverse morphisms.

<sup>1</sup> We rely on a slightly different but equivalent definition, in which the Presburger formula actually specifies a relation on the number of times each transition is taken in the run. This explains the use of Rohit Parikh’s name: a run is accepting if its Parikh image is accepted by the Presburger formula. Formal definitions appear in Section 2.



■ **Figure 2** Separations among deterministic and nondeterministic variants of PA, OCA & POCA. The arrows denote the strict subclass relation, e.g.,  $\text{DetPA} \subsetneq \text{PA}$ . All the classes not having a sequence of arrows between them are incomparable. The language in between classes belongs to the higher class but not to the lower.

**Organization of the paper.** We recall some classical notions and introduce our models in Section 2 then present some examples of POCA in Section 3. In Section 4, we state our (de)pumping lemma for POCA and rely on it to show that some languages are not expressible. We study the relationships between our classes of languages in Section 5, the closure properties of our models in Section 6, and the complexity of decision problems in Section 7.

## 2 Preliminaries

We assume the reader to be familiar with elementary automata theory.

**Sets.** We write  $\mathbb{N} = \{0, 1, 2, \dots\}$  and  $\mathbb{N}_{>0} = \{1, 2, \dots\}$ . Let  $d \in \mathbb{N}_{>0}$ . A set  $E \subseteq \mathbb{Z}^d$  is said to be *linear* if there exist vectors  $\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{Z}^d$  such that  $E = \{\mathbf{v}_0 + \sum_{i=1}^k x_i \mathbf{v}_i \mid x_1, \dots, x_k \in \mathbb{N}\}$ . A set is *semilinear* if it is a finite union of linear sets. Equivalently, a set  $E \subseteq \mathbb{Z}^d$  is semilinear if it can be represented as the set of vectors satisfying a *Presburger formula* with  $d$  free variables, that is, a first-order formula over  $(\mathbb{N}, +)$ .

**Words, languages.** We usually use  $\Sigma$  for alphabets, write  $\varepsilon$  for the empty word, and let  $\Sigma^\varepsilon$  be  $\Sigma \cup \{\varepsilon\}$ , with the understanding that  $\varepsilon \notin \Sigma$ . Any alphabet in this paper is implicitly totally ordered, so that we can speak of the  $i$ -th letter of the alphabet. This is only useful in defining the *Parikh image*  $\Phi(w)$  of a word  $w \in \Sigma^*$ : this is the vector in  $\mathbb{N}^{|\Sigma|}$  whose  $i$ -th component is the number of times the  $i$ -th letter of  $\Sigma$  appears in  $w$ .

Given two alphabets  $\Sigma, \Gamma$ , any function  $\Sigma \rightarrow \Gamma^*$  can be uniquely extended to a function  $h: \Sigma^* \rightarrow \Gamma^*$ , called a *morphism*, in such a way that  $h(\varepsilon) = \varepsilon$  and  $h(u \cdot v) = h(u) \cdot h(v)$ . For a language  $L \subseteq \Sigma^*$ , we write  $h(L)$  for  $\{h(w) \mid w \in L\}$ .

Given a language  $L \subseteq \Sigma^*$ , two words  $u, v \in \Sigma^*$  are *Myhill-Nerode equivalent* if for any  $w \in \Sigma^*$ ,  $uw \in L \Leftrightarrow vw \in L$ . This is an equivalence relation, and we write  $[u]_L$  for the set of words Myhill-Nerode equivalent to  $u$ . We will be mostly interested in  $[\varepsilon]_L$ , the set of words that can be erased from or inserted at the beginning of any word without changing its membership to  $L$ .

**Parikh one-counter automata.** A *Parikh One-Counter Automaton* (POCA)  $\mathcal{A}$  is a tuple  $(Q, q_0, \Sigma, \Delta_0, \Delta_+, F, \varphi)$  where:

- $Q$  is a finite set of states and  $q_0 \in Q$  is the initial state,
- $\Sigma$  is an alphabet,

### 30:4 Parikh One-Counter Automata

- $\Delta_0 \subseteq Q \times \Sigma^\varepsilon \times \{0, 1\} \times Q$  is a *zero-value* transition relation,
- $\Delta_+ \subseteq Q \times \Sigma^\varepsilon \times \{-1, 0, 1\} \times Q$  is a *positive* transition relation,
- $F \subseteq Q$  is a set of final states, and
- $\varphi$  is an existential Presburger formula with  $(|\Delta_0| + |\Delta_+|)$  free variables.

A *run* in  $\mathcal{A}$  is a sequence of transitions:

$$\rho = (q_1, \ell_1, b_1, q_2)(q_2, \ell_2, b_2, q_3) \cdots (q_{n-1}, \ell_{n-1}, b_{n-1}, q_n) \in (\Delta_0 \uplus \Delta_+)^*.$$

We say that  $\rho$  *starts* in  $q_1$  and *ends* in  $q_n$ . Its *trace* is the sequence of partial sums of the  $b_i$ , representing the current value of the counter:

$$\text{trace}(\rho) = \left( \sum_{i<1} b_i, \sum_{i<2} b_i, \dots, \sum_{i<n} b_i \right),$$

with the understanding that the first term of that sequence is zero. The  $i$ -th element of the trace is simply written  $\text{trace}(\rho)_i$ . The run  $\rho$  is:

- *counter-correct* if for all  $i$ ,  $\text{trace}(\rho)_i = 0 \rightarrow \rho_i \in \Delta_0$  and  $\text{trace}(\rho)_i \neq 0 \rightarrow \rho_i \in \Delta_+$ . In other words, a transition from  $\Delta_0$  is taken if the current value of the counter is 0 and one from  $\Delta_+$  if the counter is nonzero.
- *initial* if it starts in  $q_0$ , *final* if it ends in a state in  $F$ .
- *constraint-correct* if  $\Phi(\rho)$  satisfies  $\varphi$ .
- *accepting* if it is initial, final, counter-correct, and constraint-correct.

The *label* of the run  $\rho$  is the concatenation of all the  $\ell_i$  and a word  $w$  is accepted by  $\mathcal{A}$  if it is the label of an accepting run. Finally, the *language recognized* by the POCA  $\mathcal{A}$  is the set  $\mathcal{L}(\mathcal{A})$  of words accepted by it.

A POCA  $\mathcal{A}$  is said to be *deterministic* (DetPOCA) if for all states  $q$  and for both  $\Delta \in \{\Delta_0, \Delta_+\}$ :

- There are no two transitions in  $\Delta$  from  $q$  having the same label; and
- If there is a transition in  $\Delta$  from  $q$  labeled  $\varepsilon$ , there is no other transition from  $q$  in  $\Delta$ .

**Parikh Automata & One-Counter Automata.** These two models are restrictions of POCA:

- A *Parikh Automaton* (PA) is a POCA in which  $\Delta_0 \subseteq Q \times \Sigma^\varepsilon \times \{0\} \times Q$ . In this case,  $\Delta_+$  is not useful, so we simply omit it from the tuple representation and do not write the 0 update of  $\Delta_0$ . We use DetPA for the deterministic variant.
- A *One-Counter Automaton* (OCA) is a POCA in which  $\varphi$  is a tautology; we then omit it from the tuple representation. We use DetOCA for the deterministic variant.

### 3 Examples

We start with a few examples of POCA languages, which will help in clarifying the relationship of POCA with PA and OCA.

► **Example 1.** Let  $T = \{a^n b^n \mid n > 0\}$  and define:

$$L = \{u_1 u_2 \cdots u_m c^m \mid m > 0 \wedge u_1, u_2, \dots, u_m \in T\}.$$

This language can be shown to be unrecognizable by PA or OCA. Intuitively, it is not a PA language since it has unbounded sequences of words from  $T$ , each of which necessitating an equality check, and it is not an OCA language since  $m$  and  $n$ , respectively the number of  $c$

and the number of  $a, b$ , require two separate counters. However,  $L$  is recognizable using a POCA. Indeed, the counter can be used to check that each subword in  $a^+b^+$  is in  $T$  and the semilinear constraint can be used to check that the number of words  $u_i$  is exactly the number of  $c$ . In fact, this process is deterministic, hence  $L$  is a DetPOCA language.

► **Example 2.** Let again  $T = \{a^n b^n \# \mid n > 0\}$  and  $K = \{a^n b^n c^n \# \mid n > 0\}$  and define:

$$L_T = \{a, b, \#\}^* T \{a, b, \#\}^* \quad \text{and} \quad L_K = \{a, b, c, \#\}^* K \{a, b, c, \#\}^*.$$

Using a single counter,  $L_T$  can be deterministically recognized: every time we switch from  $b$  to  $a$ , we can reset the counter in search for a subword in  $T$ . Thus  $L_T$  is a DetOCA (hence DetPOCA) language. However, it can be shown that  $L_T$  is a PA language, but not a DetPA language with tools from [4]. As for  $L_K$ , since  $K$  cannot be recognized by an OCA, we need to use the semilinear constraint to recognize it: a PA for  $L_K$  would simply guess the position of the word in  $K$ , read the next  $a^+b^+c^+$  using a separate part of the automaton, then check that there were as many  $a, b$ , and  $c$  in this subword using the semilinear constraint. Thus  $L_K$  is a PA (hence POCA) language. We can show, however, that it is not a DetPOCA language.

► **Example 3.** The previous example seems to indicate that a language that is expressible with both PA and OCA can only be deterministic if it is expressible with a DetPA or DetOCA. This is not the case. Take for instance  $B = \{a^i b^j c^k \mid i \neq j \vee j \neq k\}$  and define:

$$L = \{a^n \# u_1 \# u_2 \# \cdots \# u_m \mid m > n > 0 \wedge u_1, u_2, \dots, u_m \in a^* b^* c^* \wedge u_n \in B\},$$

in words, the number of  $a$  at the beginning indicates where to find a block in  $B$ . This language is in OCA and PA but in neither DetOCA nor DetPA. It is, however, in DetPOCA: one would use the counter to find  $u_n$  (with  $n$  the number of  $a$  at the beginning), and check that  $u_n \in B$  using the semilinear constraint.

## 4 Pumping lemmas: Statements and Applications

Although pumping lemmas abound for context-free languages, and thus for OCA, there is no known technique in the PA world that takes any PA language  $L$  and a long enough word in  $L$ , and creates a word of different length in  $L$ . As mentioned in the introduction, the main expressiveness lemma for PA relies on the fact that any cycle taken twice can be moved around without changing membership. Relying on this and by carefully analyzing the behavior of the counter on long enough runs, we can show an expressiveness lemma reminiscent of a pumping property:

► **Lemma 4** (Depumping lemma). *Let  $L \subseteq \Sigma^*$  be a POCA (resp. DetPOCA) language. For any infinite language  $K \subseteq [\varepsilon]_L$  and  $N \in \mathbb{N}$ , there are words  $u, (u_i)_{i \leq n}, v, (v_j)_{j \leq m}$ , with  $m, n \geq N$ , such that all of the following hold:*

1.  $x = (u_1 u)(u_2 u) \cdots (u_n u) \cdot (v_1 v)(v_2 v) \cdots (v_{m-1} v)v_m$  is in  $K$ ,
2.  $uv \neq \varepsilon$ ,
3. There exist  $w_1, w_2 \in \Sigma^*$  such that, letting  $x' = u_1 \cdots u_n v_1 \cdots v_m$ , it holds that  $w_1 x' w_2$  is in  $L$  (resp. in  $[\varepsilon]_L$ ).

**Proof sketch.** Let  $\mathcal{A}$  be a POCA recognizing  $L$ . Now, let  $\{x_1, x_2, \dots\} \subseteq [\varepsilon]_L$  be an infinite set of words such that  $\text{length}(x_{i+1}) > \text{length}(x_i) > i$  for all  $i \geq 1$ . Since  $x_i \in [\varepsilon]_L$ , for all  $i \geq 1$ , we know that for any  $k$ , there is a run  $\rho_k$  of  $\mathcal{A}$  on  $x_1 \cdots x_k$  which can be extended to an accepting run. Write  $\rho_k$  as  $\pi_1 \cdots \pi_k$  with each  $\pi_i$  being the subrun corresponding to the subword  $x_i$ . (Note that  $\pi_i$  may be different for each value of  $k$ .)

For a word  $w$ , we say a (sub)run of  $\mathcal{A}$  on  $w$  that starts and ends in the same state is a  $w$ -cycle. Since the  $x_i$  are increasing in length, so are the  $\pi_i$ . If we take a large enough  $k \in \mathbb{N}$ , the pigeonhole principle will ensure the existence of a nonempty word  $u$  and an index  $i \in \mathbb{N}$  such that  $u$ -cycles appear more than  $N$  times in  $\pi_i$ . Note that it would suffice to argue that we can shift these  $u$ -cycles to the other subruns  $\pi_j$  with  $j \neq i$  while preserving the validity of the run. (The reason we want to shift cycles rather than just remove them is because we want to preserve the Parikh image.) In this case,  $w_1$  and  $w_2$  would be the words labelling the runs  $\pi'_1 \cdots \pi'_{i-1}$  and  $\pi'_{i+1} \cdots \pi'_p \sigma$  where:  $p$  is the maximal index  $j$  such that we shift some  $u$ -cycle to  $\pi_j$ ,  $\pi'_j$  is the run we get from  $\pi_j$  after shifting the  $u$ -cycles, and  $\sigma$  is any run such that  $\pi_1 \pi_2 \cdots \pi_p \sigma$  is accepting. The only obstruction in doing this is that, while shifting the  $u$ -cycles, we might invalidate the run by making the counter value nonzero at a zero-value transition. This is why we have to take a (possibly) larger  $k \in \mathbb{N}$  so that, again by the pigeonhole principle, we are ensured of the existence of another word  $v$  such that  $v$ -cycles can be shifted along with  $u$ -cycles to guarantee that the above does not happen. The technical aspect of the proof lies in formalising this idea. ◀

► **Remark 5.** Lemma 4 is a *depumping* lemma: it removes portions of the word  $x$  that appear often. We forego the statement for the *pumping* lemma as we will not need it, but the same proof shows that we can swap the roles of  $x$  and  $x'$  in the lemma, thus creating a longer word.

► **Remark 6.** The proof allows for some slightly stronger variations of this statement. For instance, we can assume that  $K^*$  is a set of prefixes of words in  $L$  instead of assuming that  $K \subseteq [\varepsilon]_L$ . In this case, the conclusion in the POCA case would not change and, for DetPOCA, the third conclusion would state that  $w_1 x' w_2$  is Myhill-Nerode equivalent to *some* word in  $K$ .

We now turn to examples of languages that we will show to be outside of DetPOCA and POCA. For the rest of this section, let  $\Sigma = \{a, b, c, \#\}$ . Our examples will rely on the following languages:

- $L_{ab} = \{\#a^n \#b^m \#c^m \# \mid n, m \in \mathbb{N}\}$ ,
- $L_{bc} = \{\#a^n \#b^m \#c^m \# \mid n, m \in \mathbb{N}\}$ ,
- $L_{abc} = \{\#a^n \#b^n \#c^n \# \mid n \in \mathbb{N}\} = L_{ab} \cap L_{bc}$ .

► **Proposition 7.**  $L_{ab}^* \cup L_{bc}^*$  is not recognizable by a DetPOCA.

**Proof.** We rely on Lemma 4. Assume that it is recognizable by a DetPOCA, pick  $L_{abc}$  as the language  $K$  in Lemma 4, and set  $N = 4$ . Note that we have indeed that any word of  $L_{abc}$  is Myhill-Nerode equivalent to  $\varepsilon$  in  $L_{ab}^* \cup L_{bc}^*$ . Using the notations of the lemma, we see that neither  $u$  nor  $v$  can contain  $\#$ , since they are both repeated at least 4 times in  $x$ . Thus, removing the repetitions of  $u$  and  $v$  from  $x$ , we obtain that  $x'$  is of the shape  $\#a^i \#b^j \#c^k \#$  but outside of  $L_{abc}$ ; note that  $x'$  has at least one letter from  $\{a, b, c\}$ . Assume that  $i \neq j$  (the case  $j \neq k$  is similar), and let  $z = \#a\#b\#cc\#$ , a word in  $L$ . For any words  $w_1, w_2$ , the word  $w_1 x' w_2 z$  cannot be in  $L$ , since  $x \notin L_{ab}$  and  $z \notin L_{bc}$ . This shows that  $w_1 x' w_2$  is not Myhill-Nerode equivalent to  $\varepsilon$ , a contradiction. ◀

► **Proposition 8.** Let  $B = \{\#a^i \#b^j \#c^k \# \mid i \neq j \vee j \neq k\}$  and  $C = (\#a^* \#b^* \#c^* \#)^*$ . The language  $C \cdot B \cdot C$  is not expressible with a DetPOCA.

**Proof.** Note that  $L_{abc} \subseteq [\varepsilon]_{CBC}$ : indeed, a word in  $L_{abc}$  is in  $C \setminus B$ . We can thus follow the proof of Proposition 7 and stop at the point where  $x'$  is seen to be of the shape  $\#a^i \#b^j \#c^k \#$  but not in  $L_{abc}$ . In fact,  $x' \in B$ . The conclusion of the pumping lemma now tells us that

there are words  $w_1, w_2$  such that  $w_1x'w_2 \in [\varepsilon]_{CBC}$ . Since  $\varepsilon \cdot x' \in CBC$ , we have that  $w_1x'w_2x' \in CBC$ , so certainly  $w_1x'w_2 \in C$  (recall that  $B \subseteq C$ ). But since  $x' \in B$ , we have that  $w_1x'w_2 \in CBC$  too, and thus that  $\varepsilon \in CBC$ , a contradiction. ◀

► **Remark 9.** The two previous languages happen to be expressible with POCA; let us investigate where the proof of inexpressibility breaks down when we rely on the conclusion in the POCA case of Lemma 4. In Proposition 7, we have the luxury of picking a  $z$ : this is not allowed in the POCA case, which simply guarantees that there are words  $w_1, w_2$  such that  $w_1x'w_2 \in L$ . And indeed, we only know that  $x'$  has an unbalanced number of  $a, b, c$ , leaving the possibility, that  $x' \in L_{ab} \cup L_{bc}$ , so  $w_1 = w_2 = \varepsilon$  would satisfy the conclusion of the Lemma. Similarly, in Lemma 4, there is no contradiction to be gained from  $x' \in B$  on its own, which is the case when  $w_1 = w_2 = \varepsilon$  in the Lemma.

► **Proposition 10.**  $L_{abc}^* = L_{ab}^* \cap L_{bc}^*$  is not recognizable by a POCA.

**Proof.** This follows the same process as the proof of Proposition 7, up to the point where  $x'$  is seen to be of the shape  $\#a^i\#b^j\#c^k\#$  but not in  $L_{abc}$ . Certainly, then, whichever words were to be put at the beginning and end of this word, we cannot reach a word in  $L$ . ◀

## 5 Expressiveness

### 5.1 Normal forms and inclusion in logarithmic space

We first note that cycles of  $\varepsilon$ -transitions, in which a POCA would guess a large number, are not needed. Additionally, we can complete the underlying automaton of a POCA so that:

► **Theorem 11.** For any (Det)POCA language  $L$ , there is a (Det)POCA  $A$  with language  $L$  and a constant  $c \in \mathbb{N}$  such that any word  $w \in L$  is accepted by  $A$  with a run of length at most  $c|w|$  that reads the whole input. In particular, all cycles of  $\varepsilon$ -transitions in  $A$  decrease the counter.

We will assume henceforth that all of our POCA are of the above type. From this, and noting that the Parikh constraint of a POCA can be made quantifier-free by adding “equality modulo  $m$ ” predicates, we immediately get:

► **Corollary 12.** Any DetPOCA language is in the class  $L$ . Any POCA language is in the class  $NL$ .

### 5.2 Separations

We compare the expressiveness of POCA with PA and OCA, in both their deterministic and nondeterministic variants; we obtain the following separations:

- **Theorem 13.** ■ *DetPA and DetOCA are strictly less expressive than DetPOCA;*  
 ■ *PA and OCA are strictly less expressive than POCA;*  
 ■ *The expressive power of DetPOCA is incomparable with that of PA and OCA.*

**Proof.** The inclusions in the first two statements are immediate and strictness was mentioned in Example 1. This example also shows that DetPOCA is not included in either PA or OCA.

What is left to show is that there are PA and OCA languages that are not in DetPOCA. The language  $CBC$  of Proposition 8 is such a language: it is not in DetPOCA, but it can be expressed with both a PA and an OCA (one simply guesses when the  $B$  word occurs and, within this word, guesses and checks which of the options  $i \neq j$  or  $j \neq k$  holds). ◀

In addition, we have exhibited (Propositions 7 and 8) languages expressible with POCA but not DetPOCA. Hence:

► **Theorem 14.** *DetPOCA are strictly less expressive than POCA.*

### 5.3 Rendering the OCA or Parikh part useless in a POCA

The wide disparity of expressive power between POCA and its base models OCA and Parikh automata stands in sharp contrast with the intuition that the two counting features of POCA are orthogonal. In this section, we explore how we can essentially “saturate” the abilities of one of the base models, in such a way that it cannot contribute meaningfully to recognizing a language. This is reminiscent of the work of [18], in which it is shown that if the shuffle of a nonregular context-free language and another language  $T$  is still context-free, then  $T$  has to be regular. In other words, the nonregular context-free language “saturates” the stack. See also [15] for a related notion of *simplest nonregular context-free language*.

Recall that the *shuffle* of two words  $u \sqcup v$  is the set  $\{u_1v_1 \cdots u_kv_k \mid u_i, v_i \in \Sigma^* \wedge u = u_1 \cdots u_k \wedge v = v_1 \cdots v_k\}$ . The shuffle of two languages is the set of shuffles of words from each language.

► **Theorem 15 (One-counter-stripping).** *Let  $L' \subseteq \Sigma^*$  and  $a, b \notin \Sigma$ . If  $L' \sqcup \{a^n b^n \mid n \geq 0\}^*$  is a (Det)POCA language, then  $L'$  is a (Det)PA language. The converse holds.*

**Proof sketch.** Let the POCA accepting  $L$  be  $\mathcal{A}$ . To prove the theorem above, we give a procedure of producing the intended Parikh automaton  $\mathcal{A}'$  accepting  $L'$ , using  $\mathcal{A}$  as a black box. The main idea is that  $\mathcal{A}'$  simulates  $\mathcal{A}$ , but along the accepting runs in  $\mathcal{A}'$ , the counter is only used in a bounded way that can be encoded in the state space; hence we can get rid of the counter while accepting  $L'$ . We formalize this idea as follows:

Let  $C, C_\varepsilon$  be the number of simple cycles with nonempty underlying word, and simple  $\varepsilon$ -cycles in  $\mathcal{A}$ . Let  $C' = C + C_\varepsilon + 1$ . Define a function  $\text{pad} : \Sigma^* \rightarrow (\Sigma \uplus \{a, b\})^*$  such that for any word  $w = c_1 \cdots c_n \in \Sigma^*$ ,  $\text{pad}(w)$  is of the form  $\text{pad}(w) = c_1 \cdot (a^{|Q|} b^{|Q|})^{C'} \cdots c_n \cdot (a^{|Q|} b^{|Q|})^{C'}$ . Note that, if  $w \in L'$ , then  $\text{pad}(w) \in L$ . We extend the function on the language  $L'$  naturally:  $\text{pad}(L) = \{\text{pad}(x) \mid x \in L\} \subset L'$ . The following lemma then holds.

► **Lemma 16.** *Let  $\text{Reach} = \{n \mid \exists (x \cdot y) \in \text{pad}(L'), \text{ and } q_f \in F, (q_0, 0) \xrightarrow{x} (q, n) \xrightarrow{y} q_f\}$  be the set of all counter values appearing in any accepting run in  $\mathcal{A}$  on reading a word from  $\text{pad}(L')$ . The set  $\text{Reach}$  is bounded, i.e., there exists a bound  $B \in \mathbb{N}$  such that, for every  $n \in \text{Reach}, n \leq B$ .*

Using the above lemma, we can already outline the procedure to construct the desired PA  $\mathcal{A}'$  such that  $L(\mathcal{A}') = L'$ . Let the POCA  $\mathcal{A}$  be of the form  $(Q, q_0, \Sigma \uplus \{a, b\}, \Delta, F, \varphi)$ . Then the PA  $\mathcal{A}'$  is of the form  $(Q', q'_0, \Sigma, \Delta', F', \varphi')$  such that,

- $Q' = Q \times \{0, 1, \dots, B\} \uplus \{r\}$ ,
- $q'_0 = (q_0, 0)$  and  $F' = F \times \{0, 1, \dots, B\}$ ,
- for every run of the form  $(q, i) \xrightarrow{l \cdot (a^{|Q|} b^{|Q|})^{C'}} (q', j) \in \Delta^*$ , where  $l \in \Sigma, q, q' \in Q$ , and  $i \leq B$ 
  - (i). if  $j \leq B$ , then  $((q, i) \xrightarrow{l} (q', j)) \in \Delta'$
  - (ii). otherwise  $((q, i) \xrightarrow{l} r) \in \Delta'$ .

Note that, for every configuration  $(q, i)$  in  $\mathcal{A}$  and  $l \in \Sigma$ , the number of runs on the word  $l \cdot (a^{|Q|} b^{|Q|})^{C'}$  is finite as  $\mathcal{A}$  does not contain any nonnegative  $\varepsilon$ -cycle.

- The construction of  $\Delta'$  from  $\Delta$  imposes a linear function  $f : \mathbb{Z}^{\Delta'} \rightarrow \mathbb{Z}^{\Delta}$ . Let the Parikh constraint for  $\mathcal{A}$  be  $\varphi$  with  $|\Delta|$  free variables. Then we define  $\varphi'$  with  $|\Delta'|$  free variables such that, a vector  $\mathbf{v} \in \mathbb{Z}^{\Delta'} \models \varphi'$  if and only if  $f(\mathbf{v}) \models \varphi$ .

It is easy to check that  $\mathcal{A}'$  accept  $w$  if and only if  $\mathcal{A}$  accepts  $\text{pad}(w)$ . This implies  $\mathcal{A}'$  recognises  $L'$ . The above procedure also preserves determinacy, i.e., if  $\mathcal{A}$  is deterministic, then  $\mathcal{A}'$  is also deterministic. ◀

Similarly, we expect that some operation would render the Parikh constraint useless. Specifically, assume that for some language  $L$ , the language  $(L\#)^*$  is recognized by a POCA  $A$ . Consider a long word  $w$  in  $L\#$ ; in the accepting run for it, we could move (pairs of) cycles *after* the  $\#$  symbol. Repeating this, we obtain a subword of  $w$  of constant length, appearing before  $\#$ , which must also be in  $L$ . Hence to recognize  $L$ , one only needs to simulate  $A$  for a counter-correct run and find the subword of constant length to check for constraint-correctness. The Parikh constraint can thus be hardcoded, and we thus conjecture:

► **Conjecture 17.** *Let  $L \subseteq \Sigma^*$  be a language and  $\# \notin L$ . If  $(L\#)^*$  is a POCA (resp. DetPOCA) language, then  $L$  is an OCA (resp. DetOCA) language.*

## 6 Closure Properties

We study in this section the closure properties of the POCA classes. We start with positive closure properties and then move to nonclosure claims. The results of this section are summarized in Figure 1, in the Introduction.

### 6.1 Positive closure properties

► **Theorem 18.** *The class of languages recognized by DetPOCA is closed under complement, inverse morphisms, and intersection/union with regular languages.*

**Proof.**

**Complement.** When there is no counter, i.e., in the DetPA case, this is fairly straightforward: the complement of a DetPA is the union of

1. the complement of the language of the underlying automaton; and
2. the language of the DetPA with the semilinear constraint negated.

For DetPOCA, this approach has multiple caveats: as we will see, DetPOCA is not closed under union, moreover, we need to take the complement of the underlying *DetOCA* which is slightly more technical. Let us sketch how to overcome these limitations. Consider a DetPOCA  $\mathcal{A}$ . We follow the standard first steps for complementing deterministic pushdown automata (e.g., [13, Chapter 10.2]):

1. Ensure that the automaton reads the input word in its entirety;
2. Mark as final any state that can reach a final state following only  $\varepsilon$ -transitions (possibly zeroing the counter along the way).

The language of  $\mathcal{A}$  is unchanged and a word is rejected iff after reading it, we reach a nonfinal state. Call  $F$  the set of final states at this point. Next, note that given the Parikh image of a run starting in the initial state, one can find, in first-order logic, which state is the last one in the run (this is the one state that has more incoming transitions taken than outgoing ones). Let us define a DetPOCA  $\mathcal{B}$  as  $\mathcal{A}$ , but with *all* states final. In addition, the constraint formula of  $\mathcal{B}$  accepts if either the last state of the run is *not* in  $F$  or if it is and the constraint

## 30:10 Parikh One-Counter Automata

formula of  $\mathcal{A}$  rejected the Parikh image. A word is accepted by  $\mathcal{B}$  if either the underlying DetOCA of  $\mathcal{A}$  rejected it or if it did accept it but the constraint set of  $\mathcal{A}$  was rejecting it. This is precisely the complement of  $\mathcal{L}(\mathcal{A})$ , concluding this proof.

**Intersection/union with regular languages, inverse morphism.** These are standard. ◀

► **Theorem 19.** *The class of languages recognized by POCA is closed under union, concatenation, morphisms, inverse morphisms, and intersection with regular languages.*

**Proof.**

**Concatenation.** This follows the classical construction for regular languages. The only complication is that the counter has to be reset after the jump (and the Parikh constraint has to undergo variable renaming as they speak about disjoint sets of variables arising from the original automata). This can be achieved using  $\varepsilon$ -transitions with decrements on the counter.

**Other closures.** These follow standard arguments. ◀

### 6.2 Nonclosure properties

► **Theorem 20.** *The class of languages recognized by DetPOCA is not closed under union, intersection, concatenation with regular languages, and morphisms.*

**Proof.**

**Union, intersection, concatenation with regular languages.** This is covered by Propositions 7, 8, and 10, respectively, noting that  $L_{ab}^*$ ,  $L_{bc}^*$ ,  $B$ , and  $C$  (using the notation therein) are all expressible using DetPOCA.

**Morphisms.** This is immediate from nonclosure under union. Indeed, take any two DetPOCA languages  $L_1, L_2$  over the same alphabet  $\Sigma$  and let  $\Gamma$  be a disjoint alphabet of the same size as  $\Sigma$ . Let  $h: \Gamma^* \rightarrow \Sigma^*$  be any bijective morphism. Certainly,  $T = L_1 \cup h^{-1}(L_2)$  is recognizable with a DetPOCA, since one can decide which language to test for by reading the first letter. Extending  $h$  so that it is the identity over  $\Sigma$ , we have, however, that  $h(T) = L_1 \cup L_2$ . Thus if DetPOCA were closed under morphism, it would be closed under union, a contradiction. ◀

► **Theorem 21.** *The class of languages recognized by POCA is not closed under intersection and complement.*

**Proof.** For intersection, this is the contents of Proposition 10. Nonclosure under complement is immediate from closure under union but not under intersection. ◀

## 7 Decision Problems

### 7.1 Classical decision problems

In this section, we study the computational complexity of some classical decision problems for automata: given one or two automata  $\mathcal{A}$  and  $\mathcal{B}$ , *nonemptiness* asks whether  $\mathcal{L}(\mathcal{A}) \neq \emptyset$ , *universality* whether  $\mathcal{L}(\mathcal{A}) = \Sigma^*$ , *inclusion* whether  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ , and *equivalence* whether  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ . From the known results listed in Figure 1, we immediately get:

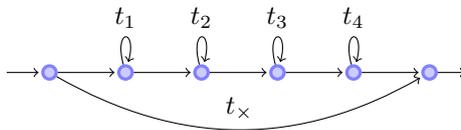
► **Corollary 22.** *Universality, inclusion, and equivalence are undecidable for POCA. Inclusion is undecidable for DetPOCA.*

► **Theorem 23.** *Nonemptiness is NP-complete for DetPOCA and POCA. Universality is coNP-complete for DetPOCA.*

**Proof.** For nonemptiness, hardness comes from the same property for DetPA and PA. In [7, Proposition III.2], this is shown by encoding instances of the SUBSETSUM problem into an instance of the nonemptiness problem for DetPA. NP-membership for nonemptiness for POCA can be shown in a way similar to [7, Proposition III.2]: we can construct an existential Presburger formula  $\varphi_\rho$  whose models form the set of Parikh images of accepting runs of the underlying OCA of a given POCA. The formula  $\varphi_\rho$  can be obtained in polynomial time using a construction from [22, Theorem 4].<sup>2</sup> We can then check whether  $\varphi \wedge \varphi_\rho$  is satisfiable, which is the case if and only if the language of the given POCA is nonempty. Since satisfiability of existential Presburger formulas is in NP (see, e.g., [10] and references therein), this concludes the proof.

For universality of DetPOCA, this is a direct consequence of the (effective!) closure of DetPOCA under complement (Theorem 18) and the previous discussion. ◀

► **Remark 24.** NP-hardness of nonemptiness has only little to do with the hardness of solving the constraint formula itself or from encoding numbers in that formula in binary. Indeed, the constraint formula obtained in the reduction from SUBSETSUM can be made quantifier-free and without constants besides 1, in which case checking that a tuple satisfies the formula is easy (in L). Let us make this reduction more explicit to see this. We build a partial PA (the counter is not needed) that either “takes” a number from the instance set or does not, in the sense that for each number  $n$  in the set, there will be a transition  $t$  that is either taken  $n$  times or 0 times. We present the construction through an example: The following partial PA will select whether  $n = 13$  gets into the candidate subset:



The constraint formula would assert (writing  $t_i$  for the number of times  $t_i$  is taken):

$$t_x = 1 \vee ((t_1 = 1) \wedge (t_2 = t_1 + t_1 + 1) \wedge (t_3 = t_2 + t_2) \wedge (t_4 = t_3 + t_3 + 1)),$$

corresponding to the binary encoding of 13, that is, 1101. Transition  $t_4$  is thus taken exactly 13 times or not at all, and the reduction is concluded by summing these selection transitions and checking if they are unequal to the target value.

► **Corollary 25.** *It is coNP-complete to decide, given a POCA  $\mathcal{A}$  and a regular language  $R$  as an NFA, whether  $\mathcal{L}(\mathcal{A}) \subseteq R$ .*

► **Open Question 26.** *Is equivalence decidable for DetPOCA? We conjecture that it is.*

## 7.2 Parametric decision problems

In the field of formal verification, computational models represent so-called *reactive systems* that communicate with and evolve based on their surrounding environment. To formalize the varying conditions provided by the environment, the models receive *parameters*, usually

<sup>2</sup> See [12] for a construction that fixes a small mistake in the proof of that theorem.

integer valued variables [1]. In [3], it is shown that reachability in parametric timed automata with two clocks correspond to parametric emptiness problem for certain classes of one-counter machines. In [8] and [19], the authors showed that parametric emptiness for OCA (“for any parameter value, the OCA is nonempty”) is decidable, where the parameters may appear on the updates of the counter. We study that problem for POCA; there are two natural places where parameters could appear: the counter updates and within the constraint formula.

- **Definition 27.** A parametric POCA is a tuple  $\mathcal{A} = (Q, q_0, \Sigma, X, \Delta_0, \Delta_+, F, \varphi)$  where:
- $Q, q_0, \Sigma,$  and  $F$  are as in POCA,
  - $X = \{x_1, \dots, x_n\}$  is a finite set of parameters that will take integer values,
  - $\Delta_0$  and  $\Delta_+$  are as in POCA but can also include tuples of the form  $(p, a, +x, q)$  and  $(p, a, -x, q)$  with  $a \in \Sigma^\varepsilon$  and  $x \in X$ ,
  - $\varphi$  has  $d + n$  free variables, where  $d = |\Delta_0| + |\Delta_+|$ , corresponding to the Parikh image of the run and the valuation of the parameters in  $X$ .

Given a valuation  $\mu: X \rightarrow \mathbb{N}$ ,  $\mathcal{A}$  induces a POCA  $\mathcal{A}_\mu$  with well-defined runs, language, etc. A POCA with parametric updates is a parametric POCA in which  $\varphi$  does not have occurrence of the parameters and a POCA with parametric constraint is a parametric POCA in which  $\Delta_0$  and  $\Delta_+$  only have nonparametric transitions.

Given a parametric POCA  $\mathcal{A}$  with parameter set  $X$ , the *parametric universal nonemptiness problem*, PUNE for short, asks whether it holds that, for all  $\mu: X \rightarrow \mathbb{N}$ , we have  $\mathcal{L}(\mathcal{A}_\mu) \neq \emptyset$ .

► **Theorem 28.** The PUNE problem for POCA with parametric updates is undecidable. It is decidable and complete for  $\text{coNEXP}^{\text{EXP}}$  for POCA with parametric constraint.

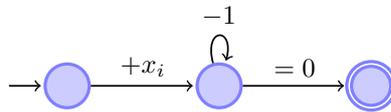
**Proof.**

**Parametric updates.** We present a reduction from Hilbert’s tenth Problem to the PUNE problem. Recall that Hilbert’s Tenth Problem asks, given a polynomial with integer coefficients, if it has a positive integer solution.

Let  $P(x_1, \dots, x_n)$  be such a polynomial and write  $P = c_1M_1 + \dots + c_kM_k$  with each  $c_i$  in  $\mathbb{Z}$  and each  $M_i$  a monomial with coefficient 1 (e.g.,  $x_1x_2^2$ ). We construct a POCA  $\mathcal{A}$  with parametric updates over the parameter set  $\{x_1, \dots, x_n\}$  that *evaluates*  $P$ . This is in the following sense: there are transitions  $t_1, \dots, t_k$  of  $\mathcal{A}$  such that for any valuation  $\mu$  of the parameters, there is a unique accepting run  $\rho$  in  $\mathcal{A}_\mu$ , and, writing  $|\rho|_{t_i}$  for the number of times  $t_i$  occurs in  $\rho$ :

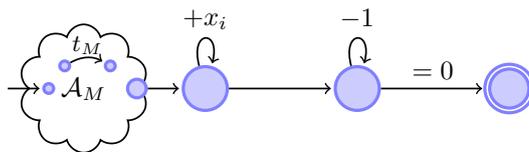
$$c_1|\rho|_{t_1} + \dots + c_k|\rho|_{t_k} = P(\mu(x_1), \dots, \mu(x_n)).$$

We start with the simplest case:  $P = x_i$ . Consider the following OCA (the labels are not important, so we assume that each transition has a unique label and do not write it):



Here, our transition  $t$  that evaluates to  $P(\mu(x_i))$  is simply the self-loop: if the run is counter-correct, this loop must have been taken  $\mu(x_i)$  times. Note that accepting runs end with a counter value of zero and that this POCA has a single final state – these are properties we will keep throughout this construction.

Next, assume  $P = Mx_i$  with  $M$  a monomial with coefficient 1. We assume that we have built a POCA  $\mathcal{A}_M$  with a transition  $t_M$  that is taken  $M(x_1, \dots, x_n)$  times on accepting runs. We then build the following POCA for  $P$ :



As constraint, we combine the Parikh constraint of  $\mathcal{A}_M$  with the statement that the  $+x_i$  loop should be taken the same number of times as  $t_M$ ; consequently, for any accepting run  $\rho$  of this POCA with valuation  $\mu$ , the  $-1$  loop is taken  $|\rho|_{t_M} \mu(x_i)$  times, which is  $M(\mu(x_1), \dots, \mu(x_n)) \mu(x_i)$  by hypothesis. This  $-1$  loop is thus the transition that evaluates to  $P(\mu(x_1), \dots, \mu(x_n))$ .

For the general case, we can chain together our POCA for each monomial one after the other, and obtain our claimed POCA for any polynomial. The constraint formula can then compute the exact value of  $P(\mu(x_1), \dots, \mu(x_n))$  and accept if it is nonzero. Thus, there is no positive integer solution to  $P$  iff for any valuation  $\mu$ ,  $\mathcal{A}_\mu$  has a nonempty language.

**Parametric constraint.** This follows the same proof as decidability in the nonparametric case, but results in a formula of Presburger arithmetic with one alternation starting with  $\forall$ . Validity of such sentences is complete for the class mentioned in the statement of the theorem [9]. ◀

## 8 Conclusion

In the long tradition of combining computational means to obtain expressive models (e.g., [20]), we have equipped one-counter automata with a mechanism to count events *globally*. This mechanism, namely constraining the Parikh image of runs to fall within a semilinear set, always enables recognizing non-context-free languages (e.g.,  $\{a^n b^n c^n \mid n > 0\}$ ) while still preserving the decidability of emptiness for models with effective semilinear Parikh images. However, studying the expressiveness of the combined model is surprisingly difficult: techniques that apply to the original model usually do not preserve satisfaction of the semilinear constraint.

Here, we have obtained expressiveness lemmas that allowed us to study the closure properties and the class relationships of the models at play. In particular, we have shown that there are languages expressible with a combination of deterministic one-counter automata and semilinear constraint that cannot be obtained by any of the underlying mechanisms: the whole is greater than the sum of its parts.

We underline research directions stemming from this work:

- We left open two main questions: 1. Is equivalence decidable for DetPOCA? We conjecture that a refinement of the algorithms for DetOCA ([21, 2]) will lead to a positive answer. 2. Is it true that if  $L' = (L\#)^*$  is a POCA language, then  $L'$  doesn't use the semilinear constraint in any meaningful way, so that  $L$  is an OCA language?
- The undecidability of the parametric universality nonemptiness problem (Theorem 28) is rather unfortunate. Indeed, we had originally expected that this problem would be a natural automata counterpart of the validity of sentences in a fragment of Presburger arithmetic with divisibility called BIL (see [19, Conclusion]). Such sentences are naturally translated to POCA with parametric updates in such a way that validity corresponds to parametric nonemptiness, but alas, validity of the BIL fragment is decidable while the PUNE problem for POCA with parametric updates is not. For context, an elegant connection between another fragment of Presburger arithmetic with divisibility and OCA with parametric updates was established in [11]: the validity problem of the former is interreducible with the nonemptiness problem of the latter via nondeterministic polynomial-time reductions. This leaves open the problem of finding such a correspondence for BIL.

- A reviewer asked the following relevant question: Can a POCA for an OCA language be more succinct than the equivalent OCA (and similarly for a POCA for a PA language)? Some examples come to mind:  $\{w \in \{a, b\}^* \mid |w|_a = 18|w|_b\}$  requires an OCA with at least 18 states, but can be done with a PA with one state. A more interesting class of languages to study this trade-off is that of languages expressible with an OCA but not by a PA, or conversely.

---

## References

- 1 Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601. ACM, 1993. doi:10.1145/167088.167242.
- 2 Stanislav Böhm, Stefan Göller, and Petr Jancar. Equivalence of deterministic one-counter automata is NL-complete. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 131–140. ACM, 2013. doi:10.1145/2488608.2488626.
- 3 Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017. doi:10.1016/j.ic.2016.07.011.
- 4 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine Parikh automata. *RAIRO – Theoretical Informatics and Applications*, 46(04):511–545, 2012. doi:10.1051/ita/2012013.
- 5 Michaël Cadilhac, Andreas Krebs, and Pierre McKenzie. The algebraic theory of parikh automata. *Theory of Computing Systems*, 62(5):1241–1268, 2017. doi:10.1007/s00224-017-9817-2.
- 6 Ehsan Chiniforooshan, Mark Daley, Oscar H. Ibarra, Lila Kari, and Shinnosuke Seki. One-reversal counter machines and multihead automata: Revisited. In *Current Trends in Theory and Practice of Computer Science*, volume 6543 of *LNCS*, pages 166–177. Springer, 2011.
- 7 Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 329–340. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.39.
- 8 Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010. doi:10.1007/978-3-642-14162-1\_48.
- 9 Christoph Haase. Subclasses of presburger arithmetic and the weak EXP hierarchy. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 – 18, 2014*, pages 47:1–47:10. ACM, 2014. doi:10.1145/2603088.2603092.
- 10 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. doi:10.1145/3242953.3242964.
- 11 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 – Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8\_25.
- 12 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification – 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2012. doi:10.1007/978-3-642-31424-7\_22.

- 13 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
- 14 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.
- 15 Petr Jancar and Jiří Šíma. The simplest non-regular deterministic context-free language. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPICs*, pages 63:1–63:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.63.
- 16 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *International Colloquium on Automata, Languages and Programming*, volume 2719 of *LNCS*, pages 681–696. Springer-Verlag, 2003. doi:10.1007/3-540-45061-0\_54.
- 17 Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2005. doi:10.1007/978-3-540-32033-3\_23.
- 18 Michel Latteux and Grzegorz Rozenberg. Commutative one-counter languages are regular. *J. Comput. Syst. Sci.*, 29(1):54–57, 1984. doi:10.1016/0022-0000(84)90013-8.
- 19 Guillermo A. Pérez and Ritam Raha. Revisiting parameter synthesis for one-counter automata. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPICs*, pages 33:1–33:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.33.
- 20 Yann-Joachim Ringard. Mustard watches: An integrated approach to time and food. Technical report, Équipe de Logique Mathématique, Paris 7, 1990. URL: <https://girard.perso.math.cnrs.fr/mustard/article.html>.
- 21 Leslie G. Valiant and Michael S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340–350, 1975. doi:10.1016/S0022-0000(75)80005-5.
- 22 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005. doi:10.1007/11532231\_25.



# Modification Problems Toward Proper (Helly) Circular-Arc Graphs

Yixin Cao  

School of Computer Science and Engineering, Central South University, Changsha, China  
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Hanchun Yuan

School of Computer Science and Engineering, Central South University, Changsha, China

Jianxin Wang

School of Computer Science and Engineering, Central South University, Changsha, China

---

## Abstract

We present a  $9^k \cdot n^{O(1)}$ -time algorithm for the proper circular-arc vertex deletion problem, resolving an open problem of van 't Hof and Villanger [Algorithmica 2013] and Crespelle et al. [Computer Science Review 2023]. Our structural study also implies parameterized algorithms for modification problems toward proper Helly circular-arc graphs.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** proper (Helly) circular-arc graph, graph modification problem

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.31

**Related Version** *Full Version*: <https://arxiv.org/abs/2202.00854>

**Funding** Supported in part by the Hong Kong Research Grants Council (RGC) under grant 15221420 and the National Natural Science Foundation of China (NSFC) under grant 61972330.

## 1 Introduction

A graph is a *circular-arc graph* if its vertices can be assigned to arcs on a circle such that there is an edge between two vertices if and only if their corresponding arcs intersect. If none of the arcs properly contains another, then the graph is a *proper circular-arc graph*. See Figure 1 for two examples of proper circular-arc graphs. Proper circular-arc graphs “form an important subclass of the class of all claw-free graphs,” and their study has been an important step towards finding “a structural characterization of all claw-free graphs” [6]. The structures and recognition of proper circular-arc graphs have been well studied and well understood [19, 8].

Another and earlier motivation for studying (proper) circular-arc graphs is from their relation with (*proper*) *interval graphs*, i.e., intersection graphs of intervals on the real line. The *intersection graph* of a family of sets has a vertex for each set and an edge between two vertices if and only if their the sets they represent have a nonempty intersection. It is easy to see that each (proper) interval graph is a (proper) circular-arc graph, and the connection of these classes has been used in both structural and algorithmic studies of these classes. Indeed, the first linear-time recognition algorithm for proper circular-arc graphs is based on a general observation of both proper circular-arc graphs and proper interval graphs [8]. Neither graph in Figure 1 is a proper interval graph, but removing any vertex from Figure 1(a), or any vertex but  $v_5$  from Figure 1(b) leaves a proper interval graph.

Let  $\mathcal{G}$  be a hereditary (closed under taking induced subgraphs) graph class. Given a graph  $G$  and an integer  $k$ , the  $\mathcal{G}$  vertex deletion problem asks whether we can remove  $k$  vertices from  $G$  to make a graph in  $\mathcal{G}$ . It is known that the  $\mathcal{G}$  vertex deletion problem is NP-hard



© Yixin Cao, Hanchun Yuan, and Jianxin Wang;  
licensed under Creative Commons License CC-BY 4.0

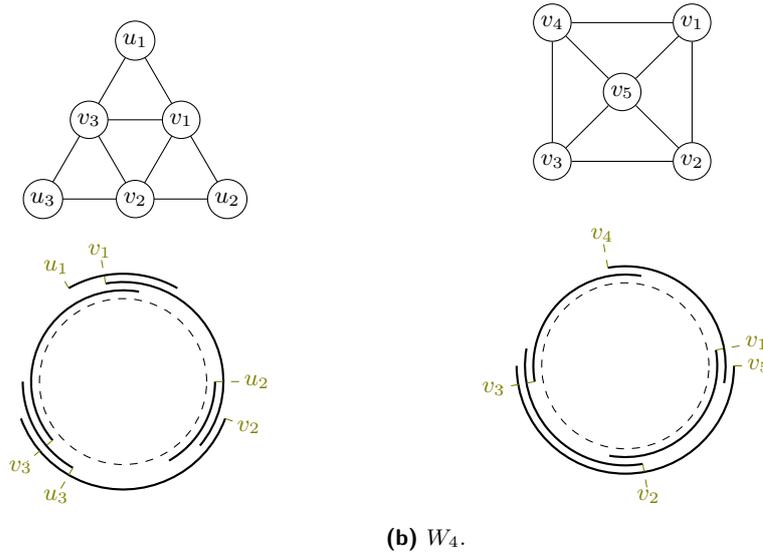
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Two proper circular-arc graphs and their arc models.

when  $\mathcal{G}$  is nontrivial (i.e., having infinite members and infinite non-members) [15]. These problems have been intensively studied in the framework of parameterized computation. Suppose that the input graph has  $n$  vertices and  $m$  edges. We say that a graph problem is *fixed-parameter tractable (FPT)* if there is an algorithm solving it in time  $f(k) \cdot n^{O(1)}$ , where  $f$  is a computable function depending only on  $k$  [9]. For example, it is well known that the proper interval vertex deletion problem is FPT [20, 3]. In the algorithm of van 't Hof and Villanger [20], the kinship between proper circular-arc graphs and proper interval graphs plays a crucial role. They showed that it suffices to destroy all the small forbidden induced subgraphs, and then the graph is already a proper circular-arc graph, on which the proper interval vertex deletion problem can be solved in linear time. They asked whether the proper circular-arc vertex deletion problem is FPT as well, and this open problem was recently raised again by Crespelle et al. [7]. We answer this question affirmatively.

► **Theorem 1.** *The proper circular-arc vertex deletion problem can be solved in time  $9^k \cdot n^{O(1)}$ .*

A major difference between the class of proper interval graphs and the class of proper circular-arc graphs is that the later class is not closed under disjoint union. This can be easily observed from their models: while we can always put intervals for two different components side by side, no such accommodation is possible for two sets of arcs if one set of them covers the whole circle. As a matter of fact, if a proper circular-arc graph is not connected, it has to be a proper interval graph. (The same remark applies to the relation between circular-arc graphs and interval graphs.)

If a proper circular-arc graph contains a hole of length at least five, then its property is quite similar to a proper interval graph. What is difficult is when a few arcs cover the whole circle in an arc model. For such a graph, it is more convenient to study its complement. Indeed, when characterizing proper circular-arc graphs, Tucker [19] actually listed the forbidden induced subgraphs of the complement class. He also observed that if the complement  $\bar{G}$  of a proper circular-arc graph  $G$  is not connected, then  $\bar{G}$  is bipartite. *Permutation graphs* are the intersection graphs of line segments between two parallel lines, and *bipartite permutation graphs* are those permutation graphs that are bipartite. Bipartite

permutation graphs are also known as proper interval bigraphs and unit interval bigraphs [14]. It is well known that a co-bipartite graph  $H$  is a proper circular-arc graph if and only if  $\overline{H}$  is a bipartite permutation graph.

Let  $(G, k)$  be an instance to the proper circular-arc vertex deletion problem, and let  $V_-$  be a solution. If  $G - V_-$  is not connected, then it is a proper interval graph; if  $G - V_-$  is not co-connected, then it is the complement of a bipartite permutation graph. We can call the algorithm of Cao [3] and the algorithm of Bożyk et al. [1] to check whether such a set  $V_-$  exists, and we are done if it does. In the rest, we may assume that  $G - V_-$  is neither a proper interval graph nor the complement of a bipartite graph, hence both connected and co-connected. For this purpose we may assume that  $G$  itself is both connected and co-connected; otherwise, there is a unique component  $C$  of  $G$  or  $\overline{G}$  such that  $V(G) \setminus V(C) \subseteq V_-$ . Either the instance is trivially FPT, when  $n = O(k)$ , or it suffices to consider the largest component of  $G$  or  $\overline{G}$ .

The algorithm proceeds as follows. We can destroy all forbidden induced subgraphs of order at most seven by branching. Now  $G$  is free of small forbidden induced subgraphs and is both connected and co-connected. Our key observation is that if  $G$  is not already a proper circular-arc graph, then  $\overline{G}$  must be bipartite. Note that any induced subgraph of a bipartite graph is bipartite, but we have assumed that  $G - V_-$  is not the complement of a bipartite graph. Therefore, we are ready to directly return “yes” or “no.”

Since the parameterized algorithm branches on a small set of vertices that intersects every solution, we can easily turn it into an approximation algorithm for the maximum proper circular-arc induced subgraph problem.

► **Theorem 2.** *There is a polynomial-time approximation algorithm of approximation ratio 9 for the minimization version of the proper circular-arc vertex deletion problem.*

Proper circular-arc graphs also arise naturally when we consider the clique graph (the intersection graph of maximal cliques of a host graph) of a circular-arc graph. The complicated structures of circular-arc graphs are mainly due to the lack of the so-called *Helly* property: every set of pairwise intersecting arcs has a common intersection. For example, neither model in Figure 1 is Helly: the set  $\{v_1, v_2, v_3\}$  in (a) and the set  $\{v_3, v_4, v_5\}$  in (b) violate the Helly property. A graph is a *Helly circular-arc graph* if it admits an arc model that is Helly. Since every interval model is Helly, all interval graphs are Helly circular-arc graphs. It is well known that the clique graph of an interval graph, with at most  $n$  maximal cliques, is a proper interval graph [13]. The same upper bound holds for the number of maximal cliques in a Helly circular-arc graph, and the clique graph of a Helly circular-arc graph is always a proper circular-arc graph [10]. Let us remark parenthetically that a circular-arc graph may have an exponential number of maximal cliques, e.g., the complement of the union of  $p$  disjoint edges, which has  $2p$  vertices, each of degree  $2p - 2$ .

The class of proper Helly circular-arc graphs is sandwiched between proper circular-arc graphs and proper interval graphs. This observation has been crucial for the algorithms for modification problems toward proper interval graphs [3]. A graph is a *proper Helly circular-arc graph* if it has an arc model that is both proper and Helly. A word of caution is worth on the definition of proper Helly circular-arc graphs. One graph might admit two arc models, one being proper and the other Helly, but no arc model that is both proper and Helly. For example, both models in Figure 1 are proper but neither is Helly, and it is not difficult to make Helly arc models for  $S_3$  and  $W_4$ , but, as the reader may easily verify, neither of them admits an arc model that is both proper and Helly. Therefore, the class of proper Helly circular-arc graphs does not contain all those graphs being both proper circular-arc graphs and Helly circular-arc graphs, but a proper subclass of it. Indeed, a proper circular-arc

graph is a proper Helly circular-arc graph if and only if it is  $\{S_3, W_4\}$ -free [16]. Another characterization of proper Helly circular-arc graphs is that they are precisely those graphs whose clique matrices have the circular-ones property for both rows and columns [17].

We then consider modification problems toward proper Helly circular-arc graphs. For this class we also consider the edge deletion and completion problems (a proof of their NP-completeness was provided in the full version). The edge deletion (resp., completion) problem asks whether we can delete (resp., add) at most  $k$  edges to a graph to make it satisfy certain properties. Again, we start by destroying all small forbidden induced subgraphs, up to six vertices. We show that a connected graph free of such induced subgraphs is already a proper Helly circular-arc graph. For the vertex deletion problem, either we remove all but one component, or we remove vertices to get a proper interval graph. The edge deletion problem is even simpler: if the graph is not connected, we cannot make it connected by deleting edges. Thus, depending on whether the graph is connected, either we are already done, or we are solving the proper interval edge deletion problem. This idea can even solve the general deletion problem that allows  $k_1$  vertex deletions and  $k_2$  edge deletions. The situation is quite different for the completion problem. We are happy if we can add at most  $k$  edges to make the input graph a proper interval graph. Otherwise, we have to make a connected proper Helly circular-arc graph. After we have dealt with all the small forbidden induced subgraphs, the only nontrivial case is when there is a large component, which contains a long hole  $H$ , and several small components. We need to “attach” these small components to vertices on  $H$ . Since these operations are local, we can find a solution by dynamic programming. Thus, all three problems are FPT, and they can be done in linear FPT time. Again, the parameterized algorithm for the vertex deletion problem can be easily turned into an approximation algorithm.

- **Theorem 3.** *For modification problems toward proper Helly circular-arc graphs, there are*
- *an  $O(6^k \cdot (m + n))$ -time algorithm for the vertex deletion problem;*
  - *an  $O(8^k \cdot (m + n))$ -time algorithm for the edge deletion problem;*
  - *an  $O(14^{k_1+k_2} \cdot (m + n))$ -time algorithm for the deletion problem; and*
  - *a  $k^{O(k)} \cdot (m + n)$ -time algorithm for the completion problem.*

*Moreover, there is an  $O(nm + n^2)$ -time approximation algorithm of approximation ratio 6 for the minimization version of the proper Helly circular-arc vertex deletion problem.*

Somewhat surprisingly, modification problems toward circular-arc graphs and its subclasses have not received sufficient attention. We hope our work will inspire more study in this direction. Apart from the two classes in the present paper, the next interesting class is the class of normal Helly circular-arc graphs, a super class of proper Helly circular-arc graphs. They have played crucial roles in solving modification problems to interval graphs [5, 2]. Also related and probably simpler are the modification problems toward unit (Helly) circular-arc graphs. It is well known that a graph is a proper interval graph if and only if it is a unit interval graph. However, there are proper (Helly) circular-arc graphs that are not unit (Helly) circular-arc graphs, e.g., the graph obtained from an even hole of length at least eight by adding edges to connect consecutive even-numbered vertices.

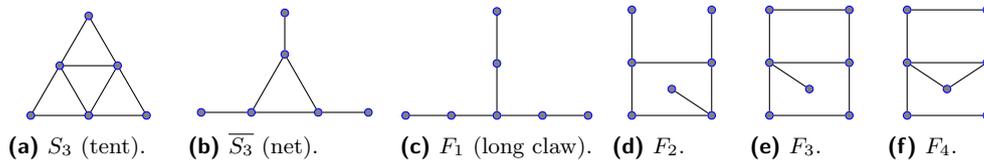
## 2 Preliminaries

All graphs discussed in this paper are undirected, simple, and finite. The vertex set and edge set of a graph  $G$  is denoted by, respectively,  $V(G)$  and  $E(G)$ . Let  $n = |V(G)|$  and  $m = |E(G)|$ . A *walk* in a graph  $G$  is a sequence of vertices and edges in the form of  $v_0$ ,

$v_0v_1, v_1, v_1v_2, \dots, v_\ell$ . Since the edges are determined by the vertices, such a walk can be denoted by  $v_0v_1 \dots v_\ell$  unambiguously. We say that this walk *connects*  $v_0$  and  $v_\ell$ , which are the *ends* of this walk, and refer to it as a  $v_0$ - $v_\ell$  walk. The *length* of a walk is the number of occurrences of edges it contains, and  $\ell$  in the previous example. A walk is *closed* if  $\ell > 1$  and  $v_0 = v_\ell$ . A walk is a *path* if all its vertices are distinct. A path is *nontrivial* if it contains at least three vertices. A closed walk of length  $\ell$  is a *cycle* if it visits precisely  $\ell$  vertices; i.e., no repeated vertices except the two ends. The length of a cycle  $C$  is denoted as  $|C|$ . For simplicity, we denote a cycle of length  $\ell$  as  $v_1v_2 \dots v_\ell$  instead of  $v_1v_2 \dots v_\ell v_1$ . The indices are understood to be modulo  $\ell$ ; e.g.,  $v_0 = v_\ell$  and  $h_{-1} = h_{\ell-1}$ . A *hole* is an induced cycle of length at least four. A walk, path, cycle, or hole is *odd* (resp., *even*) if its length is odd (resp., even). For  $\ell \geq 3$ , we use  $C_\ell$  to denote an induced cycle on  $\ell$  vertices; if we add a new vertex to a  $C_\ell$  and make it adjacent to no or all vertices on the cycle, then we end with a  $C_\ell^*$  or  $W_\ell$ , respectively.

The complement graph  $\overline{G}$  of a graph  $G$  is defined on the same vertex set  $V(G)$ , where a pair of vertices  $u$  and  $v$  is adjacent in  $\overline{G}$  if and only if  $uv \notin E(G)$ ; e.g.,  $\overline{C_5^*}$  is  $W_5$ . The graph  $\overline{C_3^*}$  is also called a *claw*. A graph  $G$  is *connected* if every pair of vertices is connected by a path, and *co-connected* if  $\overline{G}$  is connected.

A *circular-arc graph* is the intersection graph of a set of arcs on a circle. The set of arcs is called an *arc model* of this graph. In this paper, all arcs are closed. An arc model is *proper* if no arc in it properly contains another arc. A graph is a *proper circular-arc graph* if it has a proper arc model. In case there is a point of the circle avoided by all the arcs in an arc model, we can cut the circle and straighten all the arcs into line segments. Such a graph is an *interval graph*, i.e., the intersection graph of a set of closed intervals on the real line, and the set of intervals is an *interval model* of this graph. Proper interval graphs are defined analogously. Clearly, any (proper) interval model can be viewed as a (proper) arc model leaving some point uncovered, and hence all (proper) interval graphs are always (proper) circular-arc graphs.



■ **Figure 2** Some small forbidden induced graphs.

Let  $F$  be a fixed graph. We say that a graph  $G$  is  $F$ -free if  $G$  does not contain  $F$  as an induced subgraph. For a set  $\mathcal{F}$  of graphs, a graph  $G$  is  $\mathcal{F}$ -free if  $G$  is  $F$ -free for every  $F \in \mathcal{F}$ . If every  $F \in \mathcal{F}$  is minimal, i.e., not containing any  $F' \in \mathcal{F}$  as a proper induced subgraph, then  $\mathcal{F}$  comprises the (minimal) *forbidden induced subgraphs* of this class. See Figure 2 for some of the forbidden induced subgraphs considered in the present paper. We use  $S_3^*$  to denote the graph obtained by adding an isolated vertex to  $S_3$ .

► **Theorem 4** ([19]). *A graph is a proper circular-arc graph if and only if it is free of  $S_3^*, C_\ell^*$  with  $\ell \geq 4$ , as well as the complements of  $S_3, F_1, F_2, F_3, F_4, C_{2\ell+2}$ , and  $C_{2\ell-1}^*$  with  $\ell \geq 2$ .*

Neither the class of circular-arc graphs nor the class of proper circular-arc graphs is closed under taking disjoint unions. Indeed, if a (proper) circular-arc graph  $G$  is not a (proper) interval graph, then in any model of  $G$ , the union of the arcs covers the whole circle. Such a graph is necessarily connected.

## 31:6 Modification Problems Toward Proper (Helly) Circular-Arc Graphs

► **Proposition 5** (Folklore). *If a proper circular-arc graph  $G$  is not connected, then  $G$  is a proper interval graph.*

Proper circular-arc graphs have three infinite families of forbidden induced subgraphs, namely,  $\{C_\ell^* \mid \ell \geq 4\}$ ,  $\{\overline{C_{2\ell+2}} \mid \ell \geq 2\}$ , and  $\{\overline{C_{2\ell-1}^*} \mid \ell \geq 2\}$  by Theorem 4. The first of them can be ignored for connected graphs.

► **Lemma 6.** *Let  $G$  be a connected graph. If  $G$  does not contain the complement of  $C_3^*$  or the complement of  $S_3$ , then  $G$  is  $\{C_\ell^* \mid \ell \geq 5\}$ -free.*

**Proof.** Suppose for contradiction that there exist an induced cycle  $C$  and a vertex  $v$  in  $G$  with  $|C| \geq 5$  and  $V(C) \cap N(v) = \emptyset$ . Since  $G$  is connected, we can find a shortest path from  $v$  to  $C$ . Let the last three vertices on this path be  $x$ ,  $y$ , and  $z$ ; note that  $z$  is on  $C$  and  $x$  is nonadjacent to any vertex on  $C$ . We may number the vertices on  $C$  such that  $C = v_1 v_2 \cdots v_{|C|}$  and  $z = v_2$ . If  $y$  is adjacent to only  $v_2$  on  $C$ , then  $\{v_1, v_2, v_3, y\}$  induces a claw. If  $y$  is also adjacent to both  $v_1$  and  $v_3$ , then  $\{v_1, v_3, x, y\}$  induces a claw, and it is similar if  $y$  is adjacent to any three consecutive vertices on  $C$ . Otherwise,  $y$  is adjacent to precisely one of  $v_1$  and  $v_3$ . Without loss of generality, assume that  $y$  is adjacent to  $v_3$  but not  $v_1$ . Note that  $y$  is not adjacent to  $v_4$  either, and then  $\{v_1, v_2, v_3, v_4, x, y\}$  induces a copy of the complement of  $S_3$ . ◀

An arc model is *Helly* if every set of pairwise intersecting arcs has a nonempty common intersection. A circular-arc graph is *proper Helly* if it has an arc model that is both proper and Helly.

► **Theorem 7** ([17]). *A proper circular-arc graph is a proper Helly circular-arc graph if and only if it contains no  $W_4$  or  $S_3$ .*

Note that  $S_3^*$  contains  $S_3$ , while all the complements of  $F_1, F_2, F_3, F_4$ , and  $\{C_{2\ell}, C_{2\ell-1}^* \mid \ell \geq 4\}$  contain  $W_4$ . The following corollaries follow from Theorem 7, together with Theorem 4 and Lemma 6, respectively.

► **Corollary 8** ([17]). *A graph is a proper Helly circular-arc graph if and only if it contains no  $\overline{C_3^*}, S_3, \overline{S_3}, W_4, W_5, \overline{C_6}$ , or  $C_\ell^*$  for  $\ell \geq 4$ .*

► **Corollary 9.** *Let  $G$  be a connected graph. If  $G$  does not contain  $\overline{C_3^*}, C_4^*, S_3, \overline{S_3}, W_4, W_5$ , or  $\overline{C_6}$ , then  $G$  is a proper Helly circular-arc graph.*

Recall that proper interval graphs are precisely  $\{\overline{C_3^*}, S_3, \overline{S_3}, C_\ell \mid \ell \geq 4\}$ -free graphs [18, 21].

► **Corollary 10.** *Let  $G$  be a proper Helly circular-arc graph. Then  $G$  is a proper interval graph if and only if  $G$  does not contain any holes.*

The following can be viewed as a constructive version of Corollary 9.<sup>1</sup>

► **Proposition 11** (\*). *Let  $G$  be a connected graph. In  $O(m+n)$  time we can either detect an induced subgraph in  $\{\overline{C_3^*}, C_4^*, S_3, \overline{S_3}, W_4, W_5, \overline{C_6}\}$ , or build a proper and Helly arc model for  $G$ .*

A graph is a *permutation graph* if its vertices can be assigned to line segments between two parallel lines such that there is an edge between two vertices if and only if their corresponding segments intersect. The class of permutation graphs has a large number of forbidden induced subgraphs [11]. Fortunately, most of them contain an odd cycle, and thus the structures of forbidden induced subgraphs of bipartite permutation graphs are far simpler.

<sup>1</sup> Proofs of statements marked with \* are given in the full version (attached).

► **Theorem 12** ([11]). *A graph is a bipartite permutation graph if and only if it is free of  $F_1$ ,  $F_2$ ,  $F_3$ ,  $C_3$ , and  $C_\ell$  with  $\ell \geq 5$ .*

We correlate proper circular-arc graphs and bipartite permutation graphs.

► **Theorem 13** (Folklore). *The following are equivalent on a graph  $G$ :*

- i)  $G$  is a proper circular-arc graph and  $\overline{G}$  is bipartite; and
- ii)  $\overline{G}$  is a bipartite permutation graph.

The following is complement to Proposition 5 in a sense. Note that (proper) circular-arc graphs that are co-bipartite have played crucial roles in understanding these graph classes [19].

► **Proposition 14** ( $\star$ ). *Let  $G$  be a proper circular-arc graph. If  $\overline{G}$  is not connected, then  $\overline{G}$  is a bipartite permutation graph.*

### 3 Deletions to proper Helly circular-arc graphs

We first study the proper Helly circular-arc vertex deletion problem. We may assume without loss of generality that the input graph cannot be made a proper interval graph by removing  $k$  vertices. Therefore, the resulting graph after removing any  $k$ -solution is connected by Proposition 5. An FPT algorithm is immediate from Corollary 9: after destroying all the copies of  $\overline{C}_3^*$ ,  $C_4^*$ ,  $S_3$ ,  $\overline{S}_3$ ,  $W_4$ ,  $W_5$ , and  $\overline{C}_6$  in  $G$  by standard branching, we return all vertices except those in a maximum-order component. A similar (and simpler) approach works for the proper Helly circular-arc edge deletion problem. The focus of the following proof is thus on efficient implementations.

► **Theorem 15.** *The proper Helly circular-arc vertex deletion problem and the proper Helly circular-arc edge deletion problem can be solved in time  $O(6^k \cdot (m + n))$  and  $O(10^k \cdot (m + n))$ , respectively.*

**Proof.** Let  $(G, k)$  be an instance of proper Helly circular-arc vertex deletion. Our algorithm proceeds as follows. We start by calling the algorithm of Cao [3] to check whether there is a set  $V_-$  of at most  $k$  vertices such that  $G - V_-$  is a proper interval graph. If the set is found, then we return “yes.” In the rest, we look for a solution  $V_-$  such that  $G - V_-$  is *not* a proper interval graph. By Proposition 5, (note that a proper Helly circular-arc graph is a proper circular-arc graph,)  $G - V_-$  is connected.

For the general case, the algorithm solves the problem by making recursive calls to itself; we return “no” directly for a recursive call in which  $k < 0$ . For each component  $C$  of  $G$ , we call the algorithm of Proposition 11. If a subgraph induced by  $F \subseteq V(G)$  is found, then the algorithm calls itself  $|F|$  times, each with a new instance  $(G - v, k - 1)$  for some  $v \in F$ . Since we need to delete at least one vertex from  $F$ , the original instance  $(G, k)$  is a yes-instance if and only if at least one of the instances  $(G - v, k - 1)$  is a yes-instance. Now that  $G$  is free of  $\overline{C}_3^*$ ,  $C_4^*$ ,  $S_3$ ,  $\overline{S}_3$ ,  $W_4$ ,  $W_5$ , and  $\overline{C}_6$ , every component of  $G$  is a proper Helly circular-arc subgraph (Corollary 9). We find a component  $C$  of  $G$  that has the maximum order. We return “yes” if  $|V(C)| \geq n - k$ , when  $V(G) \setminus V(C)$  is a solution, or “no” otherwise. Since each of  $\overline{C}_3^*$ ,  $C_4^*$ ,  $S_3$ ,  $\overline{S}_3$ ,  $W_4$ ,  $W_5$ , and  $\overline{C}_6$  has at most 6 vertices, at most 6 recursive calls are made, all with parameter value  $k - 1$ . By Proposition 11, each recursive call can be made in  $O(m + n)$  time. Therefore, the total running time is  $O(6^k \cdot (m + n))$ .

The algorithm for the edge deletion problem is even simpler. Again, we start by calling the algorithm for proper interval edge deletion problem [3], which takes time  $O(4^k \cdot (m + n))$ . We proceed only when the answer is “no.” In the recursive calls for the general case, we always

return “no” whenever  $k$  is negative or  $G$  becomes disconnected; note that a disconnected graph cannot be made connected by edge deletions. We call the algorithm of Proposition 11, and return “yes” if  $G$  is already a proper Helly circular-arc graph. Otherwise, an induced subgraph  $F$  is found. The algorithm calls itself  $|E(F)|$  times, each with a new instance  $(G - uv, k - 1)$  for some edge  $uv$  in  $G[F]$ . By Proposition 11, each recursive call can be made in  $O(m + n)$  time. Therefore, the total running time is  $O(10^k \cdot (m + n))$ , where 10 is the number of edges in a  $W_5$ . ◀

It is straightforward to adapt an approximation algorithm for the proper Helly circular-arc vertex deletion problem from the parameterized algorithm in Theorem 15. From Theorem 15 we can easily derive an FPT algorithm for the combined deletion problem toward proper Helly circular-arc graphs, which allows  $k_1$  vertex deletions and  $k_2$  edge deletions. We can fill in the gap between the constants in Theorems 15 and 3. Only  $S_3$ ,  $W_5$ , and  $\overline{C_6}$  have more than eight edges. For an  $S_3$ , either we delete one edge between two degree-four vertices, or we have to delete both edges incident to a degree-2 vertex. For the other two cases, the ideas are similar. The details are deferred to the full version.

#### 4 Proper Helly circular-arc completion

Compared to the deletion problems, the completion problem toward proper Helly circular-arc graphs is significantly more difficult. For all the deletion problems, we can always assume that the graph is connected, and then by Corollary 9, we are only concerned with small forbidden induced subgraphs. Since adding edges may make a graph connected, we cannot assume connected input graphs for the completion problem.

Every hole in a proper Helly circular-arc graph is a dominating set of the graph, and we can be more specific on the intersection between a hole and the neighborhood of any vertex.

► **Proposition 16** ( $\star$ ). *Let  $H$  be a hole in a proper Helly circular-arc graph. Every vertex in this graph has at least two neighbors on  $H$ .*

It is well known that the maximal cliques of an interval graph can be arranged as a path. Gavril [12] showed that the maximal cliques of a Helly circular-arc graph can be arranged as a circle. This implies that a Helly circular-arc graph has a linear number of maximal cliques.

► **Theorem 17** ([12]). *A graph  $G$  is a Helly circular-arc graph if and only if its maximal cliques can be arranged as a circle so that for every vertex  $v$  in  $G$ , the maximal cliques containing  $v$  are consecutive.*

We use a *clique cycle* to denote the circular arrangement of maximal cliques specified in Theorem 17, and a *clique path* is defined analogously. In a clique path, we call the first and the last cliques *end cliques*. Note that a clique path can always be viewed as a clique cycle, while if two consecutive cliques of a clique cycle are disjoint, then it can be viewed as a clique path.

Proper interval graphs are precisely claw-free interval graphs, which can be restated as a graph is a proper interval graph if and only if it is claw-free and has a clique path. One may thus expect that a graph is a proper Helly circular-arc graph if and only if it is claw-free and has a clique cycle. As we have mentioned, however,  $S_3$  is a Helly circular-arc graph and hence has a clique cycle, but it is not a proper Helly circular-arc graph even though it is claw-free. The following statement can be directly observed from forbidden induced subgraphs of the class of proper Helly circular-arc graphs and of the class of normal Helly circular-arc graphs; see also Lin et al. [17, Theorem 9].<sup>2</sup>

<sup>2</sup> An arc model is known to be normal and Helly if no set of three or fewer arcs covers the circle [12, 4].

► **Lemma 18** ([17]). *A graph is a proper Helly circular-arc graph if and only if it is claw-free and it has an arc model in which no set of three or fewer arcs covers the circle.*

If a proper Helly circular-arc graph  $G$  is not an interval graph, then it has a hole (Corollary 10). The structure of every local part of  $G$  is very like a proper interval graph when the hole is long enough (the length at least six). With the removal of two maximal cliques with no edge in between from  $G$ , the hole is separated into two sub-paths. Since every remaining vertex is adjacent to one of the two sub-paths, the remaining graph has precisely two components.

► **Lemma 19** (\*). *Let  $G$  be a proper Helly circular-arc graph that is not an interval graph. Let  $A_1$  and  $A_2$  be two maximal cliques of  $G$  with no edge between them, and let  $B_1$  and  $B_2$  be the vertex sets of the two components of  $G - (A_1 \cup A_2)$ . Let  $G_1$  be any proper interval graph on  $B_1 \cup A_1 \cup A_2$  in which  $A_1$  and  $A_2$  are the end cliques, and  $N_{G_1}(A_i) \cap B_1 = N_G(A_i) \cap B_1$  for  $i = 1, 2$ . Replacing  $G[B_1 \cup A_1 \cup A_2]$  with  $G_1$  gives another proper Helly circular-arc graph.*

For the completion problem, we may again assume that the input graph  $G$  is free of  $\overline{C}_3^*$ ,  $C_4^*$ ,  $S_3$ ,  $\overline{S}_3$ ,  $W_4$ ,  $W_5$ , and  $\overline{C}_6$ . We are done if  $G$  is already a proper Helly circular-arc graph. In particular, this is the case when  $G$  is a proper interval graph or when  $G$  is connected (Corollary 9). Thus, we may assume that  $G$  is not connected and it is not a proper interval graph. There must be a hole in  $G$  (Corollary 10), and we add either a chord of this hole, or an edge between this hole and every vertex in other components. If there is a hole of length of no more than  $16k + 16$ , then there are only  $O(k^2)$  such choices, and we can branch on adding one of them. In the rest, every hole is longer than  $16k + 16$  (hence at least half of the vertices on  $H$  have no neighbors incident to a  $k$ -solution). Let  $H$  be such a hole, and let  $G_0$  be the component of  $G$  that contains  $H$ . Note that  $G_0$  is a proper Helly circular-arc graph (Corollary 9). After adding  $k$  or fewer edges, if the resulting graph is a proper Helly circular-arc graph, then there must be a hole of length greater than  $k$  in the subgraph induced by  $V(H)$ . Thus, for every vertex  $x$  in  $V(G) \setminus V(G_0)$ , at least two edges must be added between  $x$  and  $H$  (Proposition 16). We can return “no” if  $|V(G_0)| < n - \frac{k}{2}$ . Other components have fewer than  $k$  vertices while any hole is longer than  $16k + 16$ , and thus they are already proper interval subgraphs. They are accordingly called *small components*.

We say that a vertex  $x$  is *touched* by a solution  $E_+$  if  $x$  is an endpoint of an edge in  $E_+$ , and a set  $X$  of vertices is *touched* if at least one vertex in  $X$  is touched. All vertices in  $V(G) \setminus V(G_0)$  are touched, and we are more concerned with touched vertices in  $G_0$ .

► **Proposition 20** (\*). *Let  $E_+$  be a solution to  $G$ . If a maximal clique  $K$  of  $G$  is untouched by  $E_+$ , then  $K$  is a maximal clique of  $G + E_+$ .*

Recall that a clique cycle of a proper Helly circular-arc graph can be found in linear time [4]. We may fix a clique cycle  $\langle K_1, K_2, \dots, K_\ell \rangle$  of  $G_0$ , denoted by  $\mathcal{K}$ , and assume that  $H$  and  $\mathcal{K}$  are numbered such that no neighbor of  $v_1$  or  $v_{|H|}$  is touched, and  $\{v_1, v_{|H|}\} \subseteq K_1$ , which is untouched. Note that  $G_0$  has at least  $|H|$  maximal cliques. Since  $H$  is longer than  $16k + 16$ , few of them are touched by a  $k$ -solution  $E_+$ . By Lemma 19 and Proposition 20, these untouched maximal cliques serve as “isolators” of the modifications.

We can guess another untouched maximal clique  $K_p$  of  $G_0$  that is disjoint from and nonadjacent to  $K_1$ . By Proposition 20,  $K_1$  and  $K_p$  are both maximal cliques of a proper Helly circular-arc graph  $G + E_+$ . Since  $K_p$  is disjoint from and nonadjacent to  $K_1$ , it follows that  $G + E_+ - (K_1 \cup K_p)$  is not connected. Then  $H$  is broken into two paths in  $G - (K_1 \cup K_p)$ . Recall that every vertex in  $V(G) \setminus V(G_0)$  needs to be connected to a vertex on  $H$ . When the graph  $G + E_+$  is not a proper interval graph, every small component is connected to

## 31:10 Modification Problems Toward Proper (Helly) Circular-Arc Graphs

exactly one of the two sub-paths of  $H$ . But then the resulting graph remains connected after all vertices in  $K_1 \cup K_p$  removed, a contradiction. We can guess in  $2^k$  time to which side each small component is attached. Then we need to add edges to make two proper interval graphs. However, we cannot call the proper interval completion problem to solve this task. For example, the subgraph induced by  $\bigcup_{i=1}^p K_i$  together with the small components is already a proper interval graph. The trouble is how to make them connected while keeping  $K_1$  and  $K_p$  the end cliques of the final clique path. The same holds for the other part of the problem.

For each  $i$  with  $1 < i < |H|$ , let the maximal cliques containing  $v_i$  be  $K_{\text{from}(i)}, \dots, K_{\text{to}(i)}$ ; i.e.,  $\text{from}(i)$  and  $\text{to}(i)$  are the smallest and, respectively, the largest indices. We define  $K_{\text{to}(1)}$  and  $K_{\text{from}(|H|)}$  analogously. Let  $r$  be the number of small components, denoted as  $C_1, \dots, C_r$ . For each pair  $p, q$  of indices with  $1 \leq p < q \leq |H|$ , and each subset  $S$  of  $[1..r]$ , we check whether it is possible to add at most  $k$  edges to make  $G[\bigcup_{i=\text{to}(p)}^{\text{from}(q)} K_i \cup \bigcup_{j \in S} C_j]$  a proper interval graph, under the condition that  $K_{\text{to}(p)}$  and  $K_{\text{from}(q)}$  are the end cliques and remain untouched. Let  $\beta(S, p, q)$  denote the minimum cost if it is at most  $k$ , or  $\infty$ . We define  $\beta(S, p, q)$  to be  $\infty$  when  $K_{\text{to}(p)}$  and  $K_{\text{from}(q)}$  are not disjoint. Then  $\beta([1..r], 1, |H|)$  is the value we need, which can be calculated as follows.

► **Proposition 21.** *The value of  $\beta([1..r], 1, |H|)$  can be computed in  $k^{O(k)}(n+m)$  time.*

**Proof.** First, for  $a$  and  $b$  with  $a < b \leq a + 8k$  and  $S \subseteq [1..r]$ , we calculate  $\beta(S, a, b)$  as follows. From each component  $C_j$  with  $j \in S$ , we take a vertex  $x$ , guess a vertex  $v_i$  with  $a < i < b$ , and add the edge  $xv_i$ . After that, the subgraph induced by  $\bigcup_{i=\text{to}(a)}^{\text{from}(b)} K_i \cup \bigcup_{j \in S} C_j$  is connected. It will remain connected after adding edges. We then branch on adding edges to destroy induced subgraphs in  $\{\overline{C}_3^*, S_3, \overline{S}_3\}$  and holes in the subgraph induced by  $\bigcup_{i=\text{to}(a)}^{\text{from}(b)} K_i \cup \bigcup_{j \in S} C_j$ , without adding any edges incident to  $K_{\text{to}(a)}$  or  $K_{\text{from}(b)}$ .

Since  $G_0$  is a proper Helly circular-arc graph, a vertex is adjacent to at most four vertices on  $H$  (there is a claw otherwise). A solution is incident to at most  $2k$  vertices, and thus at most  $8k$  vertices on  $H$  have touched neighbors. If  $v_{b-1}$  has a touched neighbor, then for some  $i$  with  $2 \leq i \leq 8k + 2$ , the vertex  $v_{b-i}$  has no touched neighbor. For  $b - a > 8k$ , by Lemma 19, we have

$$\beta(S, a, b) = \min_{\substack{1 \leq i \leq 8k+1 \\ S' \subseteq S}} (\beta(S \setminus S', a, b-i) + \beta(S', b-i, b)). \quad (1)$$

We can then use dynamic programming to calculate  $\beta([1..r], 1, |H|)$  with (1). ◀

We are now ready to summarize the algorithm in Figure 3. The analysis is left to the full version.

### 5 Proper circular-arc vertex deletion

Since we will use properties of both the graph  $G$  and its complement, we beg the reader's attentiveness in reading this section. There are algorithms for the vertex deletion problem toward proper interval graphs and toward bipartite permutation graphs. We are henceforth focused on graphs that are both connected and co-connected. As usual, we can get rid of small forbidden induced subgraphs easily.

► **Definition 22.** *A graph is reduced if it is both connected and co-connected, and it contains no  $C_3^*, C_5^*, \overline{C}_4^*, C_6^*, S_3, \overline{S}_3^*, F_1, F_2, F_3$ , or  $F_4$ .*

- 
1. **if** there exists an induced subgraph  $X$  in  $\{\overline{C_3^*}, C_4^*, S_3, \overline{S_3}, W_4, W_5, \overline{C_6}\}$  **then**  
**branch** on adding missing edges of  $X$ ;  $\backslash\backslash$  returns “no” if  $k$  becomes negative.
  2. **if**  $G$  is a proper Helly circular-arc graph **then return** “yes”;
  3. find a hole  $H$  of  $G$  and let  $G_0$  be the component of  $G$  that contains  $H$ ;
  4. **if**  $|H| \leq 16k + 16$  **then**  
**branch** on adding chords of  $H$  or edges between  $H$  and other components;
  5. **if**  $|V(G_0)| < n - \frac{k}{2}$  **then return** “no”;
  6. **if**  $\beta([1..r], 1, |H|) \leq k$  **then return** “yes”;  
**else return** “no.”
- 

■ **Figure 3** The outline of the algorithm for the proper Helly circular-arc completion problem.

Similar to Proposition 11, one can make an algorithm for finding one of the subgraphs listed above when the input graph is not reduced. We omit details since it does not improve our main algorithm. The next lemma is complement and similar to Lemma 6.

► **Lemma 23.** *A reduced graph is  $\{C_\ell^* \mid \ell \geq 7\}$ -free.*

**Proof.** Let  $R$  be a reduced graph. Suppose for contradiction that there exist a hole  $H$  of length at least seven and a vertex that is nonadjacent to any vertex on  $H$ . Since  $R$  is connected, we can find a vertex  $x$  adjacent to  $H$ , and another vertex  $y$  that is adjacent to  $x$  but not to  $H$ . Let  $H = v_1v_2 \cdots v_{|H|}$ . We argue first that  $x$  cannot be adjacent to two consecutive vertices on  $H$ . Suppose for contradiction that  $x$  is adjacent to both  $v_1$  and  $v_2$ . Then  $v_1, v_2$ , and  $x$  form a triangle. Since  $R$  is free of  $C_3^*$ , it follows that  $x$  is adjacent to both  $v_4$  and  $v_5$ . But then  $v_1, v_2, v_4, v_5$ , and  $x$  induce a  $\overline{C_4^*}$  (note that  $v_5$  and  $v_1$  are nonadjacent because  $\ell \geq 7$ ).

Assume without loss of generality that  $x$  is adjacent to  $v_3$ . Note that  $x$  is adjacent to neither  $v_2$  nor  $v_4$ . If  $x$  is adjacent to  $v_5$  as well, then  $x$  is nonadjacent to  $v_6$ , and  $\{v_2, \dots, v_6, x, y\}$  induces an  $F_2$ . By symmetry,  $x$  cannot be adjacent to  $v_1$  either. But then  $\{v_1, \dots, v_5, x, y\}$  induces an  $F_1$ . ◀

By definition, a reduced graph is  $C_3^*$ - and  $C_5^*$ -free.

► **Corollary 24.** *A reduced graph is  $\{C_{2\ell+1}^* \mid \ell \geq 1\}$ -free.*

By Theorem 4, the definition of reduced graphs, and Lemma 6, a reduced graph is the complement of a proper circular-arc graph if and only if it does not contain any even hole of length at least eight. We will therefore be focused on long even holes. The main structural statement characterizes reduced graphs that contain long even holes.

► **Lemma 25.** *If a reduced graph contains an even hole of length at least eight, then it is bipartite.*

**Proof (sketch).** Let  $v_1v_2 \cdots v_\ell$  be an even hole with  $\ell \geq 8$  of a reduced graph  $R$ , and denote it by  $B$ . We prove the lemma with a sequence of claims.

1. No vertex on  $B$  participates in any triangle.
2. If some odd hole of  $R$  intersects  $B$ , then there exists an odd hole of  $R$  whose intersection with  $B$  is a nonempty sub-path of  $B$ .
3. If  $V(C) \cap V(B)$  is consecutive for an odd induced cycle  $C$ , then  $|V(C) \cap V(B)| \leq 4$ .
4. No odd induced cycle can intersect  $B$  (no vertex on  $B$  is contained in any odd induced cycle).

## 31:12 Modification Problems Toward Proper (Helly) Circular-Arc Graphs

Finally, we show that  $R$  does not contain any odd cycle at all. Let  $C$  be an odd induced cycle that is disjoint from  $B$ . First assume  $C = x_1x_2x_3$ . Since  $R$  is  $C_3^*$ -free, every vertex on  $B$  is adjacent to at least one vertex on  $C$ . Assume without loss of generality that  $x_1$  has the largest number of neighbors on  $B$ , and let their indices be  $i_1, i_2, \dots, i_p$ , sorted increasingly. Note that all of them have the same parity by the first claim. Since  $R$  is  $C_6$ -free,  $i_{j+1} - i_j$  is either two or at least six for all  $j = 1, \dots, p - 1$ . Since  $p \geq \lceil \frac{\ell}{3} \rceil$ , there must be three consecutive ones with differences two; assume without loss of generality, that they are  $v_1, v_3$ , and  $v_5$ . If  $\ell = 8$ , then  $x_1v_5v_6 \cdots v_8v_1$  has length six, and hence  $x_1$  must be adjacent to  $v_7$  as well; otherwise,  $x_1$  has another neighbor on  $B$  because  $p \geq \lceil \frac{\ell}{3} \rceil \geq 4$ . This neighbor forms an  $F_3$  with  $\{x_1, v_1, v_2, \dots, v_5\}$ . Now that  $|C| \geq 5$ ; let it be  $x_1x_2 \cdots x_{|C|}$ . We take  $v_i \in N(x_1) \cap V(B)$  and  $v_j \in N(x_3) \cap V(B)$ . The sub-path  $v_i v_{i+1} \cdots v_j$  forms an odd cycle with either  $x_1x_2x_3$  (when  $j - i$  is odd) or  $x_3x_4 \cdots x_{|C|}x_1$  (when  $j - i$  is even). From this odd cycle we can retrieve an induced odd cycle, which has to intersect both  $B$  and  $C$  (because both  $B$  and  $C$  themselves are induced cycles). This contradicts the fourth claim, and concludes the proof of this lemma. ◀

The following is immediate from Theorem 4, Lemma 6, and Lemma 25.

► **Corollary 26.** *If a reduced graph  $R$  is not bipartite, then  $R$  is the complement of a proper circular-arc graph.*

We are now ready to present the algorithm for the proper circular-arc vertex deletion problem in Figure 4. Let  $(G, k)$  be an instance to the problem, and we may assume without loss of generality that  $G$  does not contain any small forbidden induced subgraphs on at most seven vertices. If there is a set  $V_-$  of  $k$  vertices such that  $G - V_-$  is a proper interval graph or  $\overline{G} - V_-$  is a bipartite permutation graph, then we are done. Hence, we will look for a solution  $V_-$  such that  $G - V_-$  is both connected and co-connected (Propositions 5 and 14).

For this purpose we may assume that  $G$  itself is connected and co-connected: if  $G$  is not connected, we can work on the components of  $G$  one by one, and it is similar for  $\overline{G}$ . Thus,  $\overline{G}$  is a reduced graph. If  $\overline{G}$  is not bipartite, then  $G$  is already a proper circular-arc graph (Corollary 26). Otherwise,  $\overline{G}$  is bipartite, of which any induced subgraph of it is bipartite. In other words, if there exists a solution  $V_-$ , then  $\overline{G} - V_-$  is a bipartite permutation graph, and this has been handled already.

- 
1. **if**  $(G, k)$  is a yes-instance of proper interval vertex deletion **then**  
    **return** “yes”;
  2. **if**  $(\overline{G}, k)$  is a yes-instance of bipartite permutation vertex deletion **then**  
    **return** “yes”;  
     $\parallel$  *We’re looking for a solution  $V_-$  with both  $G - V_-$  and  $\overline{G} - V_-$  connected.*
  3. **branch** on deleting vertices of small forbidden induced subgraphs;
  4.  $\mathcal{C} \leftarrow$  maximal vertex sets that are connected and co-connected;
  5. **if**  $G[C]$  is co-bipartite for all  $C \in \mathcal{C}$  **then return** “no”;
  6.  $C \leftarrow$  a maximum set from  $\mathcal{C}$  with  $G[C]$  not co-bipartite;
  7. **if**  $|V(G) \setminus V(C)| \leq k$  **then return** “yes”;  
    **else return** “no.”
- 

■ **Figure 4** The outline of the algorithm for proper circular-arc vertex deletion.

Again, it is quite straightforward to turn this algorithm into an approximation algorithm, and the proofs for Theorems 1 and 2 are left to the full version.

## References

- 1 Łukasz Bożyk, Jan Derbisz, Tomasz Krawczyk, Jana Novotná, and Karolina Okrasa. Vertex deletion into bipartite permutation graphs. *Algorithmica*, 84(8):2271–2291, 2022. doi:10.1007/s00453-021-00923-7.
- 2 Yixin Cao. Linear recognition of almost interval graphs. In Robert Krauthgamer, editor, *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–1115. SIAM, 2016. Full version available at arXiv:1403.1515. doi:10.1137/1.9781611974331.ch77.
- 3 Yixin Cao. Unit interval editing is fixed-parameter tractable. *Information and Computation*, 253:109–126, 2017. A preliminary version appeared in ICALP 2015. doi:10.1016/j.ic.2017.01.008.
- 4 Yixin Cao, Luciano N. Grippo, and Martín D. Safe. Forbidden induced subgraphs of normal Helly circular-arc graphs: Characterization and detection. *Discrete Applied Mathematics*, 216:67–83, 2017. doi:10.1016/j.dam.2015.08.023.
- 5 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015. A preliminary version appeared in SODA 2014. doi:10.1145/2629595.
- 6 Maria Chudnovsky and Paul D. Seymour. Claw-free graphs. III. Circular interval graphs. *Journal of Combinatorial Theory, Series B*, 98(4):812–834, 2008. doi:10.1016/j.jctb.2008.03.001.
- 7 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.
- 8 Xiaotie Deng, Pavol Hell, and Jing Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM Journal on Computing*, 25(2):390–403, 1996. doi:10.1137/S0097539792269095.
- 9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Undergraduate texts in computer science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 10 Guillermo Durán and Min Chih Lin. Clique graphs of Helly circular arc graphs. *Ars Combinatoria*, 60, 2001.
- 11 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18:25–66, 1967. doi:10.1007/BF02020961.
- 12 Fănică Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357–369, 1974. doi:10.1002/net.3230040407.
- 13 Bruce Hedman. Clique graphs of time graphs. *Journal of Combinatorial Theory, Series B*, 37(3):270–278, 1984. doi:10.1016/0095-8956(84)90059-5.
- 14 Pavol Hell and Jing Huang. Interval bigraphs and circular arc graphs. *Journal of Graph Theory*, 46(4):313–327, 2004. doi:10.1002/jgt.20006.
- 15 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. Preliminary versions independently presented in STOC 1978. doi:10.1016/0022-0000(80)90060-4.
- 16 Min Chih Lin, Francisco J. Soulignac, and Jayme L. Szwarcfiter. Proper Helly circular-arc graphs. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph Theoretic Concepts in Computer Science - WG 2007*, volume 4769 of LNCS, pages 248–257. Springer, 2007. doi:10.1007/978-3-540-74839-7\_24.
- 17 Min Chih Lin, Francisco J. Soulignac, and Jayme L. Szwarcfiter. Normal Helly circular-arc graphs and its subclasses. *Discrete Applied Mathematics*, 161(7-8):1037–1059, 2013. doi:10.1016/j.dam.2012.11.005.
- 18 Fred S. Roberts. Indifference graphs. In Frank Harary, editor, *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., 1968)*, pages 139–146. Academic Press, New York, 1969.

## 31:14 Modification Problems Toward Proper (Helly) Circular-Arc Graphs

- 19 Alan C. Tucker. Structure theorems for some circular-arc graphs. *Discrete Mathematics*, 7(1-2):167–195, 1974. doi:10.1016/S0012-365X(74)80027-0.
- 20 Pim van 't Hof and Yngve Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013. doi:10.1007/s00453-012-9661-3.
- 21 Gerd Wegner. *Eigenschaften der Nerven homologisch-einfacher Familien im  $R^n$* . PhD thesis, Universität Göttingen, 1967.

# Isometric Path Complexity of Graphs

Dibyayan Chakraborty

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

Jérémie Chalopin  

Laboratoire d'Informatique et Systèmes, Aix-Marseille Université and CNRS,  
Faculté des Sciences de Luminy, F-13288 Marseille, Cedex 9, France

Florent Foucaud   

Université Clermont Auvergne, CNRS, Mines Saint-Étienne,  
Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France

Yann Vaxès 

Laboratoire d'Informatique et Systèmes, Aix-Marseille Université and CNRS,  
Faculté des Sciences de Luminy, F-13288 Marseille, Cedex 9, France

---

## Abstract

A set  $S$  of isometric paths of a graph  $G$  is “ $v$ -rooted”, where  $v$  is a vertex of  $G$ , if  $v$  is one of the end-vertices of all the isometric paths in  $S$ . The *isometric path complexity* of a graph  $G$ , denoted by  $ipco(G)$ , is the minimum integer  $k$  such that there exists a vertex  $v \in V(G)$  satisfying the following property: the vertices of any isometric path  $P$  of  $G$  can be covered by  $k$  many  $v$ -rooted isometric paths.

First, we provide an  $O(n^2m)$ -time algorithm to compute the isometric path complexity of a graph with  $n$  vertices and  $m$  edges. Then we show that the isometric path complexity remains bounded for graphs in three seemingly unrelated graph classes, namely, *hyperbolic graphs*, (*theta*, *prism*, *pyramid*)-*free graphs*, and *outerstring graphs*. Hyperbolic graphs are extensively studied in *Metric Graph Theory*. The class of (*theta*, *prism*, *pyramid*)-free graphs are extensively studied in *Structural Graph Theory*, e.g. in the context of the *Strong Perfect Graph Theorem*. The class of outerstring graphs is studied in *Geometric Graph Theory* and *Computational Geometry*. Our results also show that the distance functions of these (structurally) different graph classes are more similar than previously thought.

There is a direct algorithmic consequence of having small isometric path complexity. Specifically, using a result of Chakraborty et al. [ISAAC 2022], we show that if the isometric path complexity of a graph  $G$  is bounded by a constant  $k$ , then there exists a  $k$ -factor approximation algorithm for ISOMETRIC PATH COVER, whose objective is to cover all vertices of a graph with a minimum number of isometric paths.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Shortest paths, Isometric path complexity, Hyperbolic graphs, Truemper Configurations, Outerstring graphs, Isometric Path Cover

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.32

**Related Version** *Full Version*: <https://arxiv.org/abs/2301.00278>

**Funding** *Jérémie Chalopin*: This author was financed by the ANR projects DISTANCIA (ANR-17-CE40-0015) and DUCAT (ANR-20-CE48-0006).

*Florent Foucaud*: This author was financed by the ANR project GRALMECO (ANR-21-CE48-0004-01) and the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).

*Yann Vaxès*: This author was financed by the ANR project DISTANCIA (ANR-17-CE40-0015)

**Acknowledgements** We thank Nicolas Trotignon for suggesting us to study the class of ( $t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graphs.



© Dibyayan Chakraborty, Jérémie Chalopin, Florent Foucaud, and Yann Vaxès;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics



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

## 1 Introduction

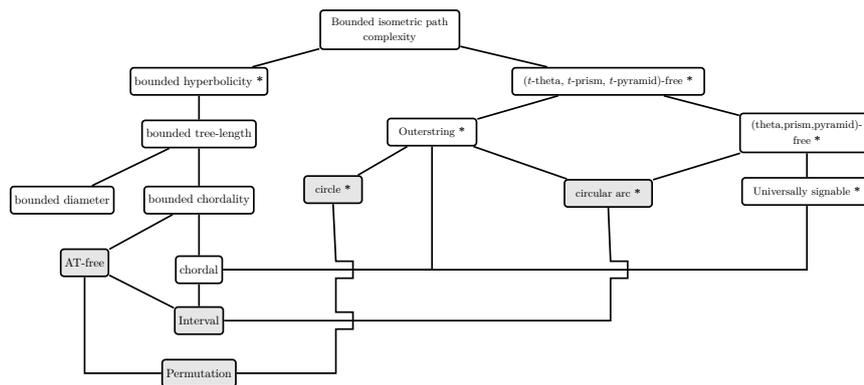
A path is *isometric* if it is a shortest path between its endpoints. An *isometric path cover* of a graph  $G$  is a set of isometric paths such that each vertex of  $G$  belongs to at least one of the paths. The *isometric path number* of  $G$  is the smallest size of an isometric path cover of  $G$ . Given a graph  $G$  and an integer  $k$ , the objective of the algorithmic problem ISOMETRIC PATH COVER is to decide if there exists an isometric path cover of cardinality at most  $k$ . ISOMETRIC PATH COVER has been introduced and studied in the context of pursuit-evasion games [2, 3]. However, until recently the algorithmic aspects of ISOMETRIC PATH COVER remained unexplored. After proving that ISOMETRIC PATH COVER remains NP-hard on *chordal graphs* (graphs without any induced cycle of length at least 4), Chakraborty et al. [7] provided constant-factor approximation algorithms for many graph classes, including *interval graphs*, chordal graphs, and more generally, graphs with bounded *treelength*. To prove the approximation ratio of their algorithm, the authors introduced a parameter called *isometric path antichain cover number* of a graph  $G$ , denoted as  $ipacc(G)$  (see Definition 6), and proved (i) when  $ipacc(G)$  is bounded by a constant, ISOMETRIC PATH COVER admits a constant-factor approximation algorithm on  $G$ ; and (ii) the isometric path antichain cover number of graphs with bounded *treelength* is bounded.

The objectives of this paper are three fold: **(A)** provide a more intuitive definition of isometric path antichain cover number; **(B)** provide a polynomial-time algorithm to compute  $ipacc(G)$ ; and **(C)** prove that it remains bounded for seemingly unrelated graph classes. Along the way, we also extend the horizon of approximability of ISOMETRIC PATH COVER. To achieve **(A)** we introduce the following new metric graph parameter, that we will show to be always equal to the isometric path antichain cover number, and whose definition is simpler.

► **Definition 1.** *Given a graph  $G$  and a vertex  $v$  of  $G$ , a set  $S$  of isometric paths of  $G$  is  $v$ -rooted if  $v$  is one of the end-vertices of all the isometric paths in  $S$ . The isometric path complexity of a graph  $G$ , denoted by  $ipco(G)$ , is the minimum integer  $k$  such that there exists a vertex  $v \in V(G)$  satisfying the following property: the vertices of any isometric path  $P$  of  $G$  can be covered by  $k$  many  $v$ -rooted isometric paths.*

A consequence of Dilworth’s theorem is that for any graph  $G$ ,  $ipacc(G) = ipco(G)$  (see Lemma 7). We will give a polynomial-time algorithm to compute  $ipco(G)$ , and therefore  $ipacc(G)$  for an arbitrary undirected graph  $G$ . This achieves **(B)**. Finally, to achieve **(C)**, we consider the following three seemingly unrelated graph classes, namely,  $\delta$ -*hyperbolic graphs*, (*theta, prism, pyramid*)-*free graphs* and *outerstring graphs*, and show that their isometric path complexity is bounded by a constant.

**$\delta$ -hyperbolic graphs.** A graph  $G$  is said to be  $\delta$ -*hyperbolic* [20] if for any four vertices  $u, v, x, y$ , the two larger of the three distance sums  $d(u, v) + d(x, y)$ ,  $d(u, x) + d(v, y)$  and  $d(u, y) + d(v, x)$  differ by at most  $2\delta$ . A graph class  $\mathcal{G}$  is *hyperbolic* if there exists a constant  $\delta$  such that every graph  $G \in \mathcal{G}$  is  $\delta$ -hyperbolic. This parameter comes from geometric group theory and was first introduced by Gromov [20] in order to study groups via their *Cayley graphs*. The hyperbolicity of a tree is 0, and in general, the hyperbolicity measures how much the distance function of a graph deviates from a tree metric. Many structurally defined graph classes like chordal graphs, *cocomparability graphs* [13], *asteroidal-triple free graphs* [14], graphs with bounded *chordality* or *treelength* are hyperbolic [8, 22]. Moreover, hyperbolicity has been found to capture important properties of several large practical graphs such as the Internet graph [25] or database relation graphs [30]. Due to its importance in discrete



■ **Figure 1** Inclusion diagram for graph classes. If a class  $A$  has an upward path to class  $B$ , then  $A$  is included in  $B$ . Constant bounds for the isometric path complexity on graph classes marked with  $*$  are contributions of this paper.

mathematics, algorithms, *metric graph theory*, researchers have studied various algorithmic aspects of hyperbolic graphs [8, 15, 10, 16]. Note that graphs with diameter 2 are hyperbolic, which may contain any graph as an induced subgraph.

**(theta, prism, pyramid)-free graphs.** A *theta* is a graph made of three vertex-disjoint induced paths  $P_1 = a \dots b$ ,  $P_2 = a \dots b$ ,  $P_3 = a \dots b$  of lengths at least 2, and such that no edges exist between the paths except the three edges incident to  $a$  and the three edges incident to  $b$ . A *pyramid* is a graph made of three induced paths  $P_1 = a \dots b_1$ ,  $P_2 = a \dots b_2$ ,  $P_3 = a \dots b_3$ , two of which have lengths at least 2, vertex-disjoint except at  $a$ , and such that  $b_1 b_2 b_3$  is a triangle and no edges exist between the paths except those of the triangle and the three edges incident to  $a$ . A *prism* is a graph made of three vertex-disjoint induced paths  $P_1 = a_1 \dots b_1$ ,  $P_2 = a_2 \dots b_2$ ,  $P_3 = a_3 \dots b_3$  of lengths at least 1, such that  $a_1 a_2 a_3$  and  $b_1 b_2 b_3$  are triangles and no edges exist between the paths except those of the two triangles. A graph  $G$  is *(theta, pyramid, prism)-free* if  $G$  does not contain any induced subgraph isomorphic to a theta, pyramid or prism. A graph is a *3-path configuration* if it is a theta, pyramid or prism. The study of 3-path configurations dates back to the works of Watkins and Meisner [31] in 1967 and plays “special roles” in the proof of the celebrated *Strong Perfect Graph Theorem* [11, 18, 27, 29]. Important graph classes like chordal graphs, *circular arc* graphs, *universally-signable* graphs [12] exclude all 3-path configurations. Popular graph classes like *perfect* graphs, *even hole-free* graphs exclude some of the 3-path configurations. Note that, *(theta, prism, pyramid)-free* graphs are not hyperbolic. To see this, consider a cycle  $C$  of order  $n$ . Clearly,  $C$  excludes all 3-path configurations and has hyperbolicity  $\Omega(n)$ .

**Outerstring graphs.** A set  $S$  of simple curves on the plane is *grounded* if there exists a horizontal line containing one endpoint of each of the curves in  $S$ . A graph  $G$  is an *outerstring* graph if there is a collection  $C$  of grounded simple curves and a bijection between  $V(G)$  and  $C$  such that two curves in  $S$  intersect if and only if the corresponding vertices are adjacent in  $G$ . The term “outerstring graph” was first used in the early 90’s [23] in the context of studying intersection graphs of simple curves on the plane. Many well-known graph classes like chordal graphs, *circular arc* graphs [19], *circle* graphs (intersection graphs of chords of a circle [17]), or cocomparability graphs [13] are also outerstring graphs and thus, motivated researchers from the *geometric graph theory* and *computational geometry* communities to study algorithmic

and structural aspects of outerstring graphs and its subclasses [4, 5, 6, 21, 24]. Note that, in general, outerstring graphs may contain a prism, pyramid or theta as an induced subgraph. Moreover, cycles of arbitrary order are outerstring graphs, implying that outerstring graphs are not hyperbolic.

It is clear from the above discussion that the classes of hyperbolic graphs, (theta, prism, pyramid)-free graphs, and outerstring graphs are pairwise incomparable (with respect to the containment relationship). We show that the isometric path complexities of all the above graph classes are small.

## 1.1 Our contributions

The main technical contributions of this paper are as follows. First we prove that the isometric path complexity can be computed in polynomial time.

► **Theorem 2.** *Given a graph  $G$  with  $n$  vertices and  $m$  edges, it is possible to compute  $ipco(G)$  in  $O(n^2m)$  time.*

Recall that, the above theorem and Lemma 7 imply that for any undirected graph  $G$ ,  $ipacc(G)$  can be computed in polynomial time. Then we show that the isometric path complexity remains bounded on hyperbolic graphs, (theta, pyramid, prism)-free graphs, and outerstring graphs. Specifically, we prove the following theorem.

► **Theorem 3.** *Let  $G$  be a graph.*

- (a) *If the hyperbolicity of  $G$  is at most  $\delta$ , then  $ipco(G) \leq 4\delta + 3$ .*
- (b) *If  $G$  is a (theta, pyramid, prism)-free graph, then  $ipco(G) \leq 71$ .*
- (c) *If  $G$  is an outerstring graph, then  $ipco(G) \leq 95$ .*

To the best of our knowledge, the isometric path complexity being bounded (by constant(s)) is the only known non-trivial property shared by any two or all three of these graph classes. Theorem 3 shows that isometric path complexity (equivalently isometric path antichain cover number), as recently introduced graph parameters, are general enough to unite these three graph classes by their metric properties. We hope that this definition will be useful for the field of metric graph theory, for example by enabling us to study (theta,prism,pyramid)-free graphs and outerstring graphs from the perspective of metric graph theory.

We provide a unified proof for Theorem 3b and 3c by proving that the isometric path complexity of ( $t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graphs [28] (see Section 4 for a definition) is bounded by a linear function of  $t$ . Due to the above theorems, we also have as corollaries that there is a polynomial-time approximation algorithm for ISOMETRIC PATH COVER with approximation ratio

- (a)  $4\delta + 3$  on  $\delta$ -hyperbolic graphs,
- (b) 73 on (theta, prism, pyramid)-free graphs,
- (c) 95 on outerstring graphs, and
- (d)  $8t + 63$  on ( $t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graphs.

To contrast with Theorem 3, we construct highly structured graphs with small *treewidth* and large isometric path complexity. A *wheel* consists of an induced cycle  $C$  of order at least 4 and a vertex  $w \notin V(C)$  adjacent to at least three vertices of  $C$ . The three path configurations introduced earlier and the wheel together are called *Truemper configurations* [29] and they are important objects of study in structural and algorithmic graph theory [1, 18].

► **Theorem 4.** For every  $k \geq 1$ ,

- (a) there exists a (pyramid, prism, wheel)-free graph  $G$  with tree-width 2, hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$  and  $\text{ipco}(G) \geq k$ .
- (b) there exists a (theta, prism, wheel)-free graph  $G$  with tree-width at most 3, hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$  and  $\text{ipco}(G) \geq k$ .
- (c) there exists a (theta, pyramid, wheel)-free graph  $G$  with hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$  and  $\text{ipco}(G) \geq k$ .

**Organisation.** In Section 2, we recall some definitions and some results. In Section 3, we present an algorithm to compute the isometric path complexity of a graph and prove Theorem 2. In Section 4, we prove Theorem 3. In Section 5, we prove Theorem 4. We conclude in Section 6. Proofs of lemma and observations marked with (\*) are provided in the main version of the paper.

## 2 Definitions and preliminary observations

In this section, we recall some definitions and some related observations. A sequence of distinct vertices forms a *path*  $P$  if any two consecutive vertices are adjacent. Whenever we fix a path  $P$  of  $G$ , we shall refer to the subgraph formed by the edges between the consecutive vertices of  $P$ . The *length* of a path  $P$ , denoted by  $|P|$ , is the number of its vertices minus one. A path is *induced* if there are no graph edges joining non-consecutive vertices. A path is *isometric* if it is a shortest path between its endpoints. For two vertices  $u, v$  of a graph  $G$ ,  $d(u, v)$  denotes the length of an isometric path between  $u$  and  $v$ .

In a directed graph, a *directed path* is a path in which all arcs are oriented in the same direction. For a path  $P$  of a graph  $G$  between two vertices  $u$  and  $v$ , the vertices  $V(P) \setminus \{u, v\}$  are *internal vertices* of  $P$ . A path between two vertices  $u$  and  $v$  is called a  $(u, v)$ -path. Similarly, we have the notions of *isometric  $(u, v)$ -path* and *induced  $(u, v)$ -path*. The interval  $I(u, v)$  between two vertices  $u$  and  $v$  consists of all vertices that belong to an isometric  $(u, v)$ -path. For a vertex  $r$  of  $G$  and a set  $S$  of vertices of  $G$ , the *distance of  $S$  from  $r$* , denoted as  $d(r, S)$ , is the minimum of the distance between any vertex of  $S$  and  $r$ . For a subgraph  $H$  of  $G$ , the *distance of  $H$  w.r.t.  $r$*  is  $d(r, V(H))$ . Formally, we have  $d(r, S) = \min\{d(r, v) : v \in S\}$  and  $d(r, H) = d(r, V(H))$ .

For a graph  $G$  and a vertex  $r \in V(G)$ , consider the following operations on  $G$ . First, remove all edges  $xy$  from  $G$  such that  $d(r, x) = d(r, y)$ . Let  $G'_r$  be the resulting graph. Then, for each edge  $e = xy \in E(G'_r)$  with  $d(r, x) = d(r, y) - 1$ , orient  $e$  from  $y$  to  $x$ . Let  $\overrightarrow{G}_r$  be the directed acyclic graph formed after applying the above operation on  $G'$ . Note that this digraph can easily be computed in linear time using a Breadth-First Search (BFS) traversal with starting vertex  $r$ . The following definition is inspired by the terminology of posets (as the graph  $\overrightarrow{G}_r$  can be seen as the Hasse diagram of a poset).

► **Definition 5.** For a graph  $G$  and a vertex  $r \in V(G)$ , two vertices  $x, y \in V(G)$  are *antichain vertices* if there are no directed paths from  $x$  to  $y$  or from  $y$  to  $x$  in  $\overrightarrow{G}_r$ . A set  $X$  of vertices of  $G$  is an *antichain set* if any two vertices in  $X$  are *antichain vertices*.

► **Definition 6** ([7]). Let  $r$  be a vertex of a graph  $G$ . For a subgraph  $H$ ,  $A_r(H)$  shall denote the maximum antichain set of  $H$  in  $\overrightarrow{G}_r$ . The *isometric path antichain cover number* of  $\overrightarrow{G}_r$ , denoted by  $\text{ipacc}(\overrightarrow{G}_r)$ , is defined as follows:

$$\text{ipacc}(\overrightarrow{G}_r) = \max \{|A_r(P)| : P \text{ is an isometric path}\}.$$

The isometric path antichain cover number of graph  $G$ , denoted as  $ipacc(G)$ , is defined as the minimum over all possible antichain covers of its associated directed acyclic graphs:

$$ipacc(G) = \min \left\{ ipacc(\vec{G}_r) : r \in V(G) \right\}.$$

For technical purposes, we also introduce the following definition. For a graph  $G$  and a vertex  $r$  of  $G$ , let  $ipco(\vec{G}_r)$  denote the minimum integer  $k$  such that any isometric path  $P$  of  $G$  can be covered by  $k$   $r$ -rooted isometric paths (The notation reflects that it is a dual notion of  $ipacc(\vec{G}_r)$ ). Using Dilworth's Theorem we prove the following important lemma.

► **Lemma 7.** For any graph  $G$  and vertex  $r$ ,  $ipco(\vec{G}_r) = ipacc(\vec{G}_r)$ . Therefore,  $ipco(G) = ipacc(G)$ .

**Proof.** Let  $r$  be a vertex of  $G$  such that any isometric path of  $G$  can be covered by  $ipco(\vec{G}_r)$   $r$ -rooted isometric paths. Let  $P$  be an arbitrary isometric path of  $G$ . Since two vertices of an antichain of  $\vec{G}_r$  cannot be covered by a single  $r$ -rooted path and  $P$  is covered by  $ipco(\vec{G}_r)$   $r$ -rooted paths, we deduce  $|A_r(P)| \leq ipco(\vec{G}_r)$ . This is true for any isometric path  $P$  of  $G$ . Hence,  $ipacc(\vec{G}_r) \leq ipco(\vec{G}_r)$ . Conversely, consider a vertex  $r \in V(G)$ . By definition of  $ipco(\vec{G}_r)$ , there is an isometric path  $P$  that cannot be covered by  $(ipco(\vec{G}_r) - 1)$   $r$ -rooted isometric paths. By Dilworth theorem,  $P$  contains an antichain of  $\vec{G}_r$  of size  $ipco(\vec{G}_r)$ . Hence  $|A_r(P)| \geq ipco(\vec{G}_r)$  and  $ipacc(\vec{G}_r) \geq ipco(\vec{G}_r)$ . The second part of the lemma follows immediately. ◀

We also recall the following theorem and proposition from [7].

► **Theorem 8 ([7]).** For a graph  $G$ , if  $ipacc(G) \leq c$ , then ISOMETRIC PATH COVER admits a polynomial-time  $c$ -approximation algorithm on  $G$ .

► **Proposition 9 ([7]).** Let  $G$  be a graph and  $r$ , an arbitrary vertex of  $G$ . Consider the directed acyclic graph  $\vec{G}_r$ , and let  $P$  be an isometric path between two vertices  $x$  and  $y$  in  $G$ . Then  $|P| \geq |d(r, x) - d(r, y)| + |A_r(P)| - 1$ .

**Proof.** Orient the edges of  $P$  from  $y$  to  $x$  in  $G$ . First, observe that  $P$  must contain a set  $E_1$  of oriented edges such that  $|E_1| = |d(r, y) - d(r, x)|$  and for any  $\vec{ab} \in E_1$ ,  $d(r, a) = d(r, b) + 1$ . Let the vertices of the largest antichain set of  $P$  in  $\vec{G}_r$ , i.e.,  $A_r(P)$ , be ordered as  $a_1, a_2, \dots, a_t$  according to their occurrence while traversing  $P$  from  $y$  to  $x$ . For  $i \in [2, t]$ , let  $P_i$  be the subpath of  $P$  between  $a_{i-1}$  and  $a_i$ . Observe that for any  $i \in [2, t]$ , since  $a_i$  and  $a_{i-1}$  are antichain vertices, there must exist an oriented edge  $\vec{b_i c_i} \in E(P_i)$  such that either  $d(r, b_i) = d(r, c_i)$  or  $d(r, b_i) = d(r, c_i) - 1$ . Let  $E_2 = \{\vec{b_i c_i}\}_{i \in [2, t]}$ . Observe that  $E_1 \cap E_2 = \emptyset$  and therefore  $|P| \geq |E_1| + |E_2| = |d(r, y) - d(r, x)| + |A_r(P)| - 1$ . ◀

### 3 Proof of Theorem 2

In this section we provide a polynomial-time algorithm to compute the isometric path complexity of a graph. Let  $G$  be a graph. In the following lemma, we provide a necessary and sufficient condition for two vertices of an isometric path to be covered by the same isometric  $r$ -rooted path in  $\vec{G}_r$  for some vertex  $r \in V(G)$ .

► **Lemma 10.** *Let  $r$  be a vertex of  $G$ . If  $P = (u = v_0, \dots, v_k = v)$  is an isometric  $(u, v)$ -path with  $d(r, u) \leq d(r, v)$  then there exists an isometric  $r$ -rooted path containing  $u, v$  in  $\overrightarrow{G}_r(P)$  if and only if  $d(v_{i+1}, r) = d(v_i, r) + 1$  for all  $i \in \{0, \dots, k - 1\}$ .*

**Proof.** If  $d(v_{i+1}, r) = d(v_i, r) + 1$  for every  $i \in \{0, \dots, k - 1\}$  then the path obtained by concatenating an isometric  $(r, u)$ -path and the path  $P$  is an isometric  $r$ -rooted  $(r, v)$ -path containing  $u, v$  in  $\overrightarrow{G}_r(P)$ . Now suppose that there exists an isometric  $r$ -rooted path containing  $u, v$  in  $\overrightarrow{G}_r(P)$ , i.e.,  $d(r, v) - d(r, u) = d(u, v)$ . Then, along any path from  $u$  to  $v$ , we need to traverse at least  $d(u, v)$  edges increasing the distance to  $r$ . Since  $P$  is an isometric  $(u, v)$ -path, it contains exactly  $d(u, v)$  edges. Hence,  $d(r, v_{i+1}) = d(r, v_i) + 1$  for every  $i \in \{0, \dots, k - 1\}$ . ◀

### 3.1 Notations and preliminary observations

We now introduce some notations that will be used to describe the algorithm and prove its correctness. Consider three vertices  $r, x, v$  of  $G$  such that  $x \neq v$ . Let  $\mathcal{P}_{\searrow}^r(x, v)$  denote the set of all isometric  $(x, v)$ -paths  $P$  containing a vertex  $u$  that is adjacent to  $v$  and satisfies  $d(r, u) = d(r, v) - 1$ . Analogously, let  $\mathcal{P}_{\rightarrow}^r(x, v)$  denote the set of all isometric  $(x, v)$ -paths  $P$  containing a vertex  $u$  that is adjacent to  $v$  and satisfies  $d(r, u) = d(r, v)$  and let  $\mathcal{P}_{\nearrow}^r(x, v)$  denote the set of all isometric  $(x, v)$ -paths  $P$  containing a vertex  $u$  that is adjacent to  $v$  and satisfies  $d(r, u) = d(r, v) + 1$ . Observe that the set of isometric  $(x, v)$ -paths is precisely  $\mathcal{P}_{\searrow}^r(x, v) \cup \mathcal{P}_{\rightarrow}^r(x, v) \cup \mathcal{P}_{\nearrow}^r(x, v)$  and that some of these sets may be empty.

Given a path  $P$ , we denote by  $|S_r(P)|$  the minimum size of a set of isometric  $r$ -rooted paths covering the vertices of  $P$ . We denote by  $\gamma_{\searrow}^r(x, v)$  and  $\beta_{\searrow}^r(x, v)$  respectively the minimum of  $|S_r(P)|$  and  $|S_r(P - \{v\})|$  over all paths  $P \in \mathcal{P}_{\searrow}^r(x, v)$ . More formally,

$$\begin{aligned} \gamma_{\searrow}^r(x, v) &= \max \{ |S_r(P)| : P \in \mathcal{P}_{\searrow}^r(x, v) \}, \\ \beta_{\searrow}^r(x, v) &= \max \{ |S_r(P - \{v\})| : P \in \mathcal{P}_{\searrow}^r(x, v) \}. \end{aligned}$$

Note that if  $\mathcal{P}_{\searrow}^r(x, v)$  is empty, we have  $\gamma_{\searrow}^r(x, v) = \beta_{\searrow}^r(x, v) = 0$ . We define similarly  $\gamma_{\nearrow}^r(x, v)$ ,  $\beta_{\nearrow}^r(x, v)$ , and  $\gamma_{\rightarrow}^r(x, v)$ :

$$\begin{aligned} \gamma_{\nearrow}^r(x, v) &= \max \{ |S_r(P)| : P \in \mathcal{P}_{\nearrow}^r(x, v) \}, \\ \beta_{\nearrow}^r(x, v) &= \max \{ |S_r(P - \{v\})| : P \in \mathcal{P}_{\nearrow}^r(x, v) \}, \\ \gamma_{\rightarrow}^r(x, v) &= \max \{ |S_r(P)| : P \in \mathcal{P}_{\rightarrow}^r(x, v) \}. \end{aligned}$$

Finally, let  $\gamma^r(x, v) = \max \{ \gamma_{\searrow}^r(x, v), \gamma_{\rightarrow}^r(x, v), \gamma_{\nearrow}^r(x, v) \}$  be the maximum of  $|S_r(P)|$  over all isometric  $(x, v)$ -paths  $P$ . In our algorithm, we will need also to consider the case where  $v = x$  as an initial case. For practical reasons, we let  $\gamma^r(x, x) = \gamma_{\searrow}^r(x, x) = \gamma_{\rightarrow}^r(x, x) = \gamma_{\nearrow}^r(x, x) = 1$  and  $\beta_{\searrow}^r(x, x) = \beta_{\nearrow}^r(x, x) = 0$ . Based on the above notations and Lemma 7, we have the following observation.

► **Observation 11.** *For any graph  $G$  and any vertex  $r$  of  $G$ , we have  $ipco(\overrightarrow{G}_r) = ipacc(\overrightarrow{G}_r) = \max_{x,v} \gamma^r(x, v)$  and  $ipco(G) = ipacc(G) = \min_r \max_{x,v} \gamma^r(x, v)$ .*

Observation 11 implies that to compute the isometric path complexity of a graph it is enough to compute the parameter  $\gamma^r(x, v)$  for all  $r, x, v \in V(G)$  in polynomial time. In the next section, we focus on achieving this goal without computing explicitly any of the sets  $\mathcal{P}_{\searrow}^r(x, v)$ ,  $\mathcal{P}_{\rightarrow}^r(x, v)$  or  $\mathcal{P}_{\nearrow}^r(x, v)$ . (Note that the size of these sets could be exponential in the number of vertices of the graph).

### 3.2 An algorithm to compute $\gamma^r(x, v)$

Throughout this section, let  $r$  and  $x$  be two fixed vertices of  $G$ . We shall call  $r$  as the “root” and  $x$  as the “source” vertex. The objective of this section is to compute the parameter  $\gamma^r(x, v)$  for all vertices  $v \in V(G)$ .

In the sequel, since we always refer to a fixed root  $r$  and source  $x$ , we omit  $r$  and  $x$  and use the shorthand  $\gamma(v)$  for  $\gamma^r(x, v)$ . We do the same with the notations  $\gamma_{\nearrow}(v)$ ,  $\gamma_{\rightarrow}(v)$ ,  $\gamma_{\searrow}(v)$ ,  $\beta_{\nearrow}(v)$ , and  $\beta_{\searrow}(v)$  that also refer to fixed vertices  $r$  and  $x$ . In the following lemmas, we shall provide explicit (recursive) formulas to compute  $\gamma_{\nearrow}(v)$ ,  $\gamma_{\rightarrow}(v)$ ,  $\gamma_{\searrow}(v)$ ,  $\beta_{\nearrow}(v)$ , and  $\beta_{\searrow}(v)$ . Using these formulas, we will show how to compute  $\gamma(v)$  for all  $v \in V(G)$  in a total of  $O(|E(G)|)$ -time.

► **Observation 12.** *If  $r$  is the root vertex,  $x$  the source vertex, and  $v$  is distinct from  $x$ , then*

$$\begin{aligned}\beta_{\searrow}(v) &= \max\{\gamma(u) : u \in I(x, v) \cap N(v); d(r, u) = d(r, v) - 1\}, \\ \beta_{\nearrow}(v) &= \max\{\gamma(u) : u \in I(x, v) \cap N(v); d(r, u) = d(r, v) + 1\}.\end{aligned}$$

► **Lemma 13 (\*)**. *If  $r$  is the root vertex,  $x$  the source vertex, and  $v$  is distinct from  $x$ , then  $\gamma_{\rightarrow}(v) = \max\{1 + \gamma(u) : u \in I(x, v) \cap N(v); d(r, u) = d(r, v)\}$ .*

► **Lemma 14 (\*)**. *If  $r$  is the root vertex,  $x$  the source vertex, and  $v$  is a vertex distinct from  $x$ , then  $\gamma_{\searrow}(v) = \max\{\max\{\gamma_{\searrow}(u), \gamma_{\rightarrow}(u), \beta_{\nearrow}(u) + 1\} : u \in I(x, v) \cap N(v); d(r, u) = d(r, v) - 1\}$*

► **Lemma 15 (\*)**. *If  $r$  is the root vertex,  $x$  the source vertex, and  $v$  is a vertex distinct from  $x$ , then  $\gamma_{\nearrow}(v) = \max\{\max\{\gamma_{\nearrow}(u), \gamma_{\rightarrow}(u), \beta_{\searrow}(u) + 1\} : u \in I(x, v) \cap N(v); d(r, u) = d(r, v) + 1\}$ .*

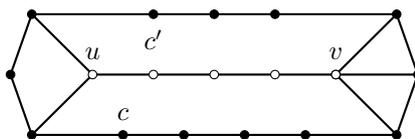
Now we provide a BFS based algorithm to compute the above parameters. Let  $r$  and  $x$  be fixed root and source vertices of  $G$ , respectively. For a vertex  $u \in V(G)$ , let  $\mathcal{D}(u) = \{\gamma(u), \gamma_{\nearrow}(u), \gamma_{\rightarrow}(u), \gamma_{\searrow}(u), \beta_{\nearrow}(u), \beta_{\searrow}(u)\}$ . Clearly, the set  $\mathcal{D}(x)$  can be computed in constant time. Now let  $X_i$  be the set of vertices at distance  $i$  from  $x$ . Clearly, the sets  $X_i$  can be computed in  $O(|E(G)|)$ -time (using a BFS) and  $X_0 = \{x\}$ . Let  $i \geq 1$  be an integer and assume that for all vertices  $u \in \bigcup_{j=0}^{i-1} X_j$ , the set  $\mathcal{D}(u)$  is already computed. Let  $v \in X_i$  be a vertex. Then due to the formulas given in Observation 12 and Lemmas 13–15, the set  $\mathcal{D}(v)$  can be computed by observing only the sets  $\mathcal{D}(u)$ ,  $u \in N(v) \cap X_{i-1}$ . Hence, for all vertices  $v \in V(G)$ , the sets  $\mathcal{D}(v)$  can be computed in a total of  $O(|E(G)|)$  time. Hence, we have the following lemma.

► **Lemma 16.** *For a root vertex  $r$  and source vertex  $x$ , for all vertices  $v \in V(G)$ , the value  $\gamma^r(x, v)$  can be computed in  $O(|E(G)|)$  time.*

We can now finish the proof of Theorem 2. Let  $G$  be a graph with  $n$  vertices and  $m$  edges. For a root vertex  $r$ , by applying Lemma 16, for every source  $x \in V(G)$ , it is possible to compute  $ipco\left(\overrightarrow{G}_r\right) = \max_{x,v} \gamma^r(x, v)$  in  $O(nm)$  time. By repeating this for every root  $r \in V(G)$ , it is possible to compute  $ipco(G) = \min_r ipco\left(\overrightarrow{G}_r\right)$  in  $O(n^2m)$  time.

## 4 Proof of Theorem 3

**First we prove Theorem 3a.** We recall the definition of Gromov products [20] and its relation with hyperbolicity. For three vertices  $r, x, y$  of a graph  $G$ , the Gromov product of  $x, y$  with respect to  $r$  is defined as  $(x|y)_r = \frac{1}{2}(d(x, r) + d(y, r) - d(x, y))$ . Then, a graph  $G$  is  $\delta$ -hyperbolic [9, 20] if and only if for any four vertices  $x, y, z, r$ , we have  $(x|y)_r \geq \min\{(x|z)_r, (y|z)_r\} - \delta$ .



■ **Figure 2** An example of a 4-fat turtle. Let  $C$  be the cycle induced by the black vertices,  $P$  be the path induced by the white vertices. Then the tuple  $(4, C, P, c, c')$  defines a 4-fat turtle.

Let  $G$  be a graph with hyperbolicity at most  $\delta$ . Due to Lemma 7, in order to prove Theorem 3a, it is enough to show that  $ipacc(G) \leq 4\delta + 3$ . Aiming for a contradiction, let  $r$  be a vertex of  $G$  and  $P$  be an isometric path such that  $|A_r(P)| \geq 4\delta + 4$ . Let  $a_1, a_2, \dots, a_{2\delta+2}, \dots, a_{4\delta+4}$  be the vertices of  $A_r(P)$  ordered as they are encountered while traversing  $P$  from one end-vertex to the other. Let  $x = a_1, z = a_{2\delta+2}, y = a_{4\delta+4}$ . Let  $Q$  denote the  $(y, z)$ -subpath of  $P$ . Observe that,  $|A_r(Q)| \geq 2\delta + 2$ . Then we have  $(x|y)_r \geq \min\{(x|z)_r, (y|z)_r\} - \delta$ . Without loss of generality, assume that  $(x|z)_r \leq (y|z)_r$ . Hence,

$$\begin{aligned} (x|y)_r &\geq (x|z)_r - \delta \\ d(x, r) + d(y, r) - d(x, y) &\geq d(x, r) + d(z, r) - d(x, z) - 2\delta \\ d(y, r) - d(x, y) &\geq d(z, r) - d(x, z) - 2\delta \\ d(y, r) - d(z, r) + 2\delta &\geq d(x, y) - d(x, z) \\ d(y, r) - d(z, r) + 2\delta &\geq d(y, z) \\ d(y, z) &\leq |d(y, r) - d(z, r)| + 2\delta. \end{aligned}$$

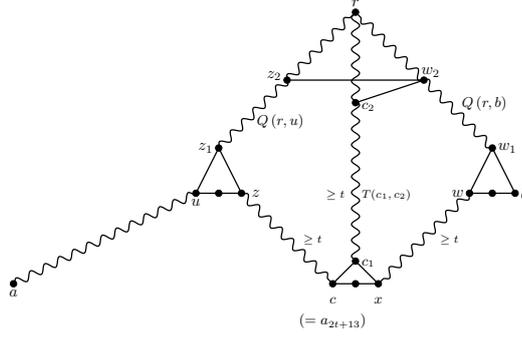
But this directly contradicts Proposition 9, which implies that  $d(y, z) \geq |d(y, r) - d(z, r)| + |A_r(Q)| - 1 \geq |d(y, r) - d(z, r)| + 2\delta + 1$ . This completes the proof of Theorem 3a.

**Now, we shall prove Theorems 3b and 3c.** First, we shall define the notions of  $t$ -theta,  $t$ -prism, and  $t$ -pyramid [28]. For an integer  $t \geq 1$ , a  $t$ -prism is a graph made of three vertex-disjoint induced paths  $P_1 = a_1 \dots b_1, P_2 = a_2 \dots b_2, P_3 = a_3 \dots b_3$  of lengths at least  $t$ , such that  $a_1 a_2 a_3$  and  $b_1 b_2 b_3$  are triangles and no edges exist between the paths except those of the two triangles. For an integer  $t \geq 1$ , a  $t$ -pyramid is a graph made of three induced paths  $P_1 = a \dots b_1, P_2 = a \dots b_2, P_3 = a \dots b_3$  of lengths at least  $t$ , two of which have lengths at least  $t + 1$ , they are pairwise vertex-disjoint except at  $a$ , such that  $b_1 b_2 b_3$  is a triangle and no edges exist between the paths except those of the triangle and the three edges incident to  $a$ . For an integer  $t \geq 1$ , a  $t$ -theta is a graph made of three internally vertex-disjoint induced paths  $P_1 = a \dots b, P_2 = a \dots b, P_3 = a \dots b$  of lengths at least  $t + 1$ , and such that no edges exist between the paths except the three edges incident to  $a$  and the three edges incident to  $b$ . A graph  $G$  is  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free if  $G$  does not contain any induced subgraph isomorphic to a  $t$ -theta,  $t$ -pyramid or  $t$ -prism. When  $t = 1$ ,  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graphs are exactly  $(\theta$ , prism, pyramid)-free graphs.

Now, we shall show that the isometric path antichain cover number of  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graphs are bounded above by a linear function on  $t$ . We shall show that, when the isometric path antichain cover number of a graph is large, the existence of a structure called “ $t$ -fat turtle” (defined below) as an induced subgraph is forced, which cannot be present in a  $((t - 1)$ -theta,  $(t - 1)$ -pyramid,  $(t - 1)$ -prism)-free graph.

► **Definition 17.** For an integer  $t \geq 1$ , a “ $t$ -fat turtle” consists of a cycle  $C$  and an induced  $(u, v)$ -path  $P$  of length at least  $t$  such that all of the following hold:

- (a)  $V(P) \cap V(C) = \emptyset$ ,



■ **Figure 3** Illustration of the notations used in the proof of Lemma 20.

- (b) For any vertex  $w \in (V(P) \setminus \{u, v\})$ ,  $N(w) \cap V(C) = \emptyset$  and both  $u$  and  $v$  have at least one neighbour in  $C$ ,
- (c) For any vertex  $w \in N(u) \cap V(C)$  and  $w' \in N(v) \cap V(C)$ , the distance between  $w$  and  $w'$  in  $C$  is at least  $t$ ,
- (d) There exist two vertices  $\{c, c'\} \subset V(C)$  and two distinct components  $C_u, C_v$  of  $C - \{c, c'\}$  such that  $N(u) \cap V(C) \subseteq V(C_u)$  and  $N(v) \cap V(C) \subseteq V(C_v)$ .

The tuple  $(t, C, P, c, c')$  defines the  $t$ -fat turtle. See Figure 2 for an example.

In the following observation, we show that any  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free graph cannot contain a  $(t + 1)$ -fat turtle as an induced subgraph.

► **Lemma 18 (\*)**. For some integer  $t \geq 1$ , let  $G$  be a graph containing a  $(t + 1)$ -fat turtle as an induced subgraph. Then  $G$  is not  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free.

In the remainder of this section, we shall prove that there exists a linear function  $f(t)$  such that if the isometric path antichain cover number of a graph is more than  $f(t)$ , then  $G$  is forced to contain a  $(t + 1)$ -fat turtle as an induced subgraph, and therefore is not  $(t$ -theta,  $t$ -pyramid,  $t$ -prism)-free. We shall use the following observation.

► **Observation 19 (\*)**. Let  $G$  be a graph,  $r$  be an arbitrary vertex,  $P$  be an isometric  $(u, v)$ -path in  $G$  and  $Q$  be a subpath of an isometric  $(v, r)$ -path in  $G$  such that one endpoint of  $Q$  is  $v$ . Let  $P'$  be the maximum  $(u, w)$ -subpath of  $P$  such that no internal vertex of  $P'$  is a neighbour of some vertex of  $Q$ . We have that  $|A_r(P')| \geq |A_r(P)| - 3$ .

► **Lemma 20**. For an integer  $t \geq 1$ , let  $G$  be a graph with  $\text{ipacc}(G) \geq 8t + 64$ . Then  $G$  has a  $(t + 1)$ -fat turtle as an induced subgraph.

**Proof.** Let  $r$  be a vertex of  $G$  such that  $\text{ipacc}(\vec{G}_r)$  is at least  $8t + 64$ . Then there exists an isometric path  $P$  such that  $|A_r(P)| \geq 8t + 64$ . Let the two endpoints of  $P$  be  $a$  and  $b$ . (See Figure 3.) Let  $u$  be a vertex of  $P$  such that  $d(r, u) = d(r, P)$ . Let  $P(a, u)$  be the  $(a, u)$ -subpath of  $P$  and  $P(b, u)$  be the  $(b, u)$ -subpath of  $P$ . Both  $P(a, u)$  and  $P(b, u)$  are isometric paths and observe that either  $|A_r(P(a, u))| \geq 4t + 32$  or  $|A_r(P(b, u))| \geq 4t + 32$ . Without loss of generality, assume that  $|A_r(P(b, u))| \geq 4t + 32$ . Let  $Q(r, b)$  be an isometric  $(b, r)$ -path in  $G$ . First observe that  $u$  is not adjacent to any vertex of  $Q(r, b)$ . Otherwise,  $d(u, b) \leq 2 + d(r, b) - d(r, u)$ , which contradicts Proposition 9. Let  $P(u, w)$  be the maximum  $(u, w)$ -subpath of  $P(b, u)$  such that no internal vertex of  $P(u, w)$  is a neighbour of  $Q(r, b)$ . Note that  $P(u, w)$  is an isometric path and  $w$  has a neighbour in  $Q(r, b)$ . Applying Observation 19, we have the following:

▷ Claim 21.  $|A_r(P(u, w))| \geq 4t + 29$ .

Let  $Q(r, u)$  be any isometric  $(u, r)$ -path of  $G$ . Observe that  $w$  is not adjacent to any vertex of  $Q(r, u)$ . Otherwise,  $d(u, w) \leq 2 + d(r, u) - d(r, w)$ , which contradicts Proposition 9. Let  $P(z, w)$  be the maximum  $(z, w)$ -subpath of  $P(u, w)$  such that no internal vertex of  $P(z, w)$  has a neighbour in  $Q(r, u)$ . Observe that  $P(z, w)$  is an isometric path, and  $z$  has a neighbour in  $Q(r, u)$ . Again applying Observation 19, we have the following:

▷ Claim 22.  $|A_r(P(z, w))| \geq 4t + 26$ .

Let  $a_1, a_2, \dots, a_k$  be the vertices of  $A_r(P(z, w))$  ordered according to their appearance while traversing  $P(z, w)$  from  $z$  to  $w$ . Due to Claim 22, we have that  $k \geq 4t + 26$ . Let  $c = a_{2t+13}$  and  $Q(r, c)$  denote an isometric  $(c, r)$ -path. Let  $T(r, c_1)$  be the maximum subpath of  $Q(r, c)$  such that no internal vertex of  $T(r, c_1)$  is adjacent to any vertex of  $P(z, w)$ . Observe that neither  $z$  nor  $w$  can be adjacent to  $c_1$  (due to Proposition 9). Moreover, if  $c_1$  is a vertex of  $P(z, w)$  then we must have  $c_1 = c$ .

▷ Claim 23 (\*). Let  $x$  be a neighbour of  $c_1$  in  $P(z, w)$ ,  $X$  be the  $(x, b)$ -subpath of  $P(u, b)$  and  $Y$  be the  $(x, u)$ -subpath of  $P(u, b)$ . Then  $|A_r(X)| \geq 2t + 11$  and  $|A_r(Y)| \geq 2t + 11$ .

Let  $T(c_1, c_2)$  be the maximum  $(c_1, c_2)$ -subpath of  $T(c_1, r)$  such that no internal vertex of  $T(c_1, c_2)$  is adjacent to a vertex of  $Q(r, b)$  or  $Q(r, u)$ . We have the following claim.

▷ Claim 24 (\*). The length of  $T(c_1, c_2)$  is at least  $t + 3$ .

The path  $T(c_1, c_2)$  forms the first ingredient to extract a  $(t + 1)$ -fat turtle. Let  $z_1$  be the neighbour of  $z$  in  $Q(r, u)$  and  $w_1$  be the neighbour of  $w$  in  $Q(r, b)$ . We have the following claim.

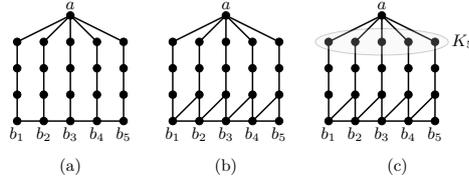
▷ Claim 25 (\*). The vertices  $w_1$  and  $z_1$  are non adjacent.

Now we shall construct a  $(w_1, z_1)$ -path as follows: Consider the maximum  $(w_1, w_2)$ -subpath, say  $T(w_1, w_2)$ , of  $Q(r, b)$  such that no internal vertex of  $T(w_1, w_2)$  has a neighbour in  $Q(r, u)$ . Similarly, consider the maximum  $(z_1, z_2)$ -subpath, say  $T(z_1, z_2)$ , of  $Q(r, u)$  such that no internal vertex of  $T(z_1, z_2)$  is a neighbour of  $w_2$ . (Note that it is possible that  $z_2 = w_2 = r$ .) Let  $T$  be the path obtained by taking the union of  $T(w_1, w_2)$  and  $T(z_1, z_2)$ . Observe that  $z_2$  must be a neighbour of  $w_2$  and  $T$  is an induced  $(w_1, z_1)$ -path. The definitions of  $T$  and  $P(z, w)$  imply that their union induces a cycle  $Z$ . Here we have the second and final ingredient to extract the  $(t + 1)$ -fat turtle.

Suppose that  $c_2$  has a neighbour in  $T$ . Let  $T'$  be the maximum subpath of  $T(c_1, c_2)$  which is vertex-disjoint from  $Z$ . (Note that if  $c_1 = c$  or  $c_2 \in \{w_2, z_2\}$  (e.g. when  $c_2 = w_2 = z_2 = r$ ),  $T(c_1, c_2)$  may share vertices with  $Z$ .) Due to Claim 24, the length of  $T'$  is at least  $t + 1$ . Let  $e_1$  and  $e_2$  be the end-vertices of  $T'$ . Observe the following.

- Each of  $e_1$  and  $e_2$  has at least one neighbour in  $Z$ .
- $Z - \{z, w\}$  contains two distinct components  $C_1, C_2$  such that for  $i \in \{1, 2\}$ ,  $N(e_i) \cap V(Z) \subseteq V(C_i)$ .
- For a vertex  $e'_1 \in N(e_1) \cap V(Z)$  and  $e'_2 \in N(e_2) \cap V(Z)$ , the distance between  $e'_1$  and  $e'_2$  is at least  $t + 1$ . This statement follows from Claim 23.

Hence, we have that the tuple  $(t + 1, Z, T', z, w)$  defines a  $(t + 1)$ -fat turtle. Now consider the case when  $c_2$  does not have a neighbour in  $T$ . By definition,  $c_2$  has at least one neighbour in  $Q(r, u)$  or  $Q(r, b)$ . Without loss of generality, assume that  $c_2$  has a neighbour  $c_3$  in  $Q(r, u)$  such that the  $(z_2, c_3)$ -subpath, say,  $T''$  of  $Q(r, u)$  has no neighbour of  $c_2$  other than  $c_3$ . Observe that the path  $T^* = (T' \cup (T'' - \{z_2\}))$  is vertex-disjoint from  $Z$  and has length at least  $t + 1$ . Let  $e_1, e_2$  be the two end-vertices of  $T^*$ . Observe the following.



■ **Figure 4** (a)  $X_4$  (b)  $Y_4$  (c)  $Z_4$ .

- Each of  $e_1$  and  $e_2$  has at least one neighbour in  $Z$ .
- $Z - \{z, w\}$  contains two distinct components  $C_1, C_2$  such that for  $i \in \{1, 2\}$ ,  $N(e_i) \cap V(Z) \subseteq V(C_i)$ .
- For a vertex  $e'_1 \in N(e_1) \cap V(Z)$  and  $e'_2 \in N(e_2) \cap V(Z)$ , the distance between  $e'_1$  and  $e'_2$  is at least  $t + 1$ . This statement follows from Claim 23.

Hence,  $(t + 1, Z, T^*, z, w)$  is a  $(t + 1)$ -fat turtle. ◀

**Proof of Theorem 3b.** Lemma 7, 18 and 20 together imply Theorem 3b.

► **Lemma 26 (\*)**. Any outerstring graph is (4-theta, 4-prism, 4-pyramid)-free.

**Proof of Theorem 3c.** Lemma 7, 18, 20, and 26 together imply Theorem 3c.

## 5 Proof of Theorem 4

We shall provide a construction for every  $k \geq 4$ , this implies the statement of Theorem 4 for any  $k \geq 1$ . First we shall prove Theorem 4a. For a fixed integer  $k \geq 4$ , first we describe the construction of a graph  $X_k$  as follows. Consider  $k + 1$  paths  $P_1, P_2, \dots, P_{k+1}$  each of length  $k$  and having a common endvertex  $a$ . For  $i \in [k + 1]$ , let the other endvertex of  $P_i$  be denoted as  $b_i$ . Moreover, for  $i \in [k + 1]$ , let the neighbours of  $a$  and  $b_i$  in  $P_i$  be denoted as  $a'_i$  and  $b'_i$ , respectively. For  $i \in [k]$ , introduce an edge between  $b_i$  and  $b_{i+1}$ . The resulting graph is denoted  $X_k$  and the special vertex  $a$  is the *apex* of  $X_k$ . See Figure 4(a). For a fixed integer  $k \geq 4$ , consider the graph  $X_k$  and for each  $i \in [k]$ , introduce an edge between  $b_i$  and  $b'_{i+1}$ . Let  $Y_k$  denote the resulting graph and the special vertex  $a$  is the *apex* of  $Y_k$ . See Figure 4(b). For a fixed integer  $k \geq 4$ , consider the graph  $Y_k$  and for each  $\{i, j\} \subseteq [k]$ , introduce an edge between  $a'_i$  and  $a'_j$ . Let  $Z_k$  denote the resulting graph and the special vertex  $a$  is the *apex* of  $Z_k$ . See Figure 4(c). We prove the following lemmas.

► **Lemma 27 (\*)**. For  $k \geq 4$ , let  $G$  be the graph constructed by taking two distinct copies of  $X_k$  and identifying the two apex vertices. Then  $G$  is a wheel-free, (pyramid, prism)-free graph with treewidth 2, hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$  and  $\text{ipacc}(G) \geq k$ .

► **Lemma 28 (\*)**. For  $k \geq 4$ , let  $G$  be the graph constructed by taking two distinct copies of  $Y_k$  and identifying the two apex vertices. Then  $G$  is a wheel-free, (theta, prism)-free graph with treewidth 3, hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$ , and  $\text{ipacc}(G) \geq k$ .

► **Lemma 29 (\*)**. For  $k \geq 4$ , let  $G$  be the graph constructed by taking two distinct copies of  $Z_k$  and identifying the two apex vertices. Then  $G$  is a wheel-free, (theta, pyramid)-free graph with hyperbolicity at least  $\lceil \frac{k}{2} \rceil - 1$  and  $\text{ipacc}(G) \geq k$ .

Lemma 7, 27, 28, 29 imply Theorem 4.

## 6 Conclusion

In this paper, we have introduced the new graph parameter *isometric path complexity*. We have shown that the isometric path complexity of a graph with  $n$  vertices and  $m$  edges can be computed in  $O(n^2m)$ -time. It would be interesting to provide a faster algorithm to compute the isometric path complexity of a graph. We have derived upper bounds on the isometric path complexity of three seemingly (structurally) different classes of graphs, namely hyperbolic graphs,  $(\theta, \text{pyramid, prism})$ -free graphs and outerstring graphs. An interesting direction of research is to generalise the properties of hyperbolic graphs or  $(\theta, \text{pyramid, prism})$ -free graphs to graphs with bounded isometric path complexity.

Note that, in our proofs we essentially show that, for any graph  $G$  that belongs to one of the above graph classes, any vertex  $v$  of  $G$ , and any isometric path  $P$  of  $G$ , the path  $P$  can be covered by a small number of  $v$ -rooted isometric paths. This implies our “choice of the root” is arbitrary. This motivates the following definition. The *strong isometric path complexity* of a graph  $G$  is the minimum integer  $k$  such that for each vertex  $v \in V(G)$  we have that the vertices of any isometric path  $P$  of  $G$  can be covered by  $k$  many  $v$ -rooted isometric paths. Our proofs imply that the strong isometric path complexity of graphs from all the graph classes addressed in this paper are bounded. We also wonder whether one can find other interesting graph classes with small (strong) isometric path complexity.

Our results imply a constant-factor approximation algorithm for ISOMETRIC PATH COVER on hyperbolic graphs,  $(\theta, \text{pyramid, prism})$ -free graphs and outerstring graphs. However, the existence of a constant-factor approximation algorithm for ISOMETRIC PATH COVER on general graphs is not known (an  $O(\log n)$ -factor approximation algorithm is designed in [26]).

---

## References

- 1 P. Aboulker, M. Chudnovsky, P. Seymour, and N. Trotignon. Wheel-free planar graphs. *European Journal of Combinatorics*, 49:57–67, 2015.
- 2 I. Abraham, C. Gavoille, A. Gupta, O. Neiman, and K. Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing*, 48(3):1120–1145, 2019.
- 3 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.
- 4 T. Biedl, A. Biniaz, and M. Derka. On the size of outer-string representations. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, 2018.
- 5 P. Bose, P. Carmi, J. M. Keil, A. Maheshwari, S. Mehrabi, D. Mondal, and M. Smid. Computing maximum independent set on outerstring graphs and their relatives. *Computational Geometry*, 103:101852, 2022.
- 6 J. Cardinal, S. Felsner, T. Miltzow, C. Tompkins, and Birgit Vogtenhuber. Intersection graphs of rays and grounded segments. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 153–166. Springer, 2017.
- 7 D. Chakraborty, A. Dailly, S. Das, F. Foucaud, H. Gahlawat, and S. K. Ghosh. Complexity and algorithms for ISOMETRIC PATH COVER on chordal graphs and beyond. In *Proceedings of the 33rd International Symposium on Algorithms and Computation, ISAAC*, volume 248 of *LIPICs*, pages 12:1–12:17, 2022.
- 8 V. Chepoi, F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Diameters, centers, and approximating trees of  $\delta$ -hyperbolic geodesic spaces and graphs. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 59–68, 2008.
- 9 V. Chepoi, F. Dragan, M. Habib, Y. Vaxès, and H. Alrasheed. Fast approximation of eccentricities and distances in hyperbolic graphs. *Journal of Graph Algorithms and Applications*, 23(2):393–433, 2019.

- 10 V. Chepoi, F. F. Dragan, and Y. Vaxes. Core congestion is inherent in hyperbolic networks. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2264–2279. SIAM, 2017.
- 11 M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of mathematics*, pages 51–229, 2006.
- 12 M. Conforti, G. Cornuéjols, A. Kapoor, and K. Vušković. Universally signable graphs. *Combinatorica*, 17(1):67–77, 1997.
- 13 D. G. Corneil, B. Dalton, and M. Habib. Ldfs-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013.
- 14 D. G. Corneil, S. Olariu, and L. Stewart. Asteroidal triple-free graphs. *SIAM Journal on Discrete Mathematics*, 10(3):399–430, 1997.
- 15 D. Coudert, A. Nusser, and L. Viennot. Enumeration of far-apart pairs by decreasing distance for faster hyperbolicity computation. *arXiv preprint*, 2021. [arXiv:2104.12523](https://arxiv.org/abs/2104.12523).
- 16 B. Das Gupta, M. Karpinski, N. Mobasheri, and F. Yahyanejad. Effect of Gromov-hyperbolicity parameter on cuts and expansions in graphs and some algorithmic implications. *Algorithmica*, 80(2):772–800, 2018.
- 17 J. Davies and R. McCarty. Circle graphs are quadratically  $\chi$ -bounded. *Bulletin of the London Mathematical Society*, 53(3):673–679, 2021.
- 18 É. Diot, M. Radovanović, N. Trotignon, and K. Vušković. The (theta, wheel)-free graphs Part I: only-prism and only-pyramid graphs. *Journal of Combinatorial Theory, Series B*, 143:123–147, 2020.
- 19 M. Francis, P. Hell, and J. Stacho. Forbidden structure characterization of circular-arc graphs and a certifying recognition algorithm. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1708–1727. SIAM, 2014.
- 20 M. Gromov. Hyperbolic groups. In *Essays in group theory*, pages 75–263. Springer, 1987.
- 21 J. M. Keil, J.S.B Mitchell, D. Pradhan, and M. Vatschelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Computational Geometry*, 60:19–25, 2017.
- 22 A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. *Algorithmica*, 72(3):758–777, 2015.
- 23 J. Kratochvíl. String graphs. I. the number of critical nonstring graphs is infinite. *Journal of Combinatorial Theory, Series B*, 52(1):53–66, 1991.
- 24 A. Rok and B. Walczak. Outerstring graphs are  $\chi$ -bounded. *SIAM Journal on Discrete Mathematics*, 33(4):2181–2199, 2019.
- 25 Y. Shavitt and T. Tankel. On the curvature of the internet and its usage for overlay construction and distance estimation. In *IEEE INFOCOM 2004*, volume 1. IEEE, 2004.
- 26 M. Thiessen and T. Gaertner. Active learning of convex halfspaces on graphs. In *Proceedings of the 35th Conference on Neural Information Processing Systems, NeurIPS 2021*, volume 34, pages 23413–23425. Curran Associates, Inc., 2021. URL: <https://proceedings.neurips.cc/paper/2021/file/c4bf1e24f3e6f92ca9dfd9a7a1a1049c-Paper.pdf>.
- 27 N. Trotignon. Perfect graphs: a survey. *arXiv preprint*, 2013. [arXiv:1301.5149](https://arxiv.org/abs/1301.5149).
- 28 N. Trotignon. Private communication, 2022.
- 29 K. Vušković. The world of hereditary graph classes viewed through truemper configurations. *Surveys in Combinatorics 2013*, 409:265, 2013.
- 30 J. A. Walter and H. Ritter. On interactive visualization of high-dimensional data using the hyperbolic plane. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 123–132, 2002.
- 31 M. E. Watkins and D. M. Mesner. Cycles and connectivity in graphs. *Canadian Journal of Mathematics*, 19:1319–1328, 1967.

# Support Size Estimation: The Power of Conditioning

Diptarka Chakraborty ✉

National University of Singapore, Singapore

Gunjan Kumar ✉

National University of Singapore, Singapore

Kuldeep S. Meel ✉

National University of Singapore, Singapore

---

## Abstract

We consider the problem of estimating the support size of a distribution  $D$ . Our investigations are pursued through the lens of distribution testing and seek to understand the power of conditional sampling (denoted as COND), wherein one is allowed to query the given distribution conditioned on an arbitrary subset  $S$ . The primary contribution of this work is to introduce a new approach to lower bounds for the COND model that relies on using powerful tools from information theory and communication complexity.

Our approach allows us to obtain surprisingly strong lower bounds for the COND model and its extensions.

- We bridge the longstanding gap between the upper bound  $O\left(\log \log n + \frac{1}{\epsilon^2}\right)$  and the lower bound  $\Omega\left(\sqrt{\log \log n}\right)$  for the COND model by providing a nearly matching lower bound. Surprisingly, we show that even if we get to know the actual probabilities along with COND samples, still  $\Omega\left(\log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$  queries are necessary.
- We obtain the first non-trivial lower bound for the COND equipped with an additional oracle that reveals the actual as well as the conditional probabilities of the samples (to the best of our knowledge, this subsumes all of the models previously studied): in particular, we demonstrate that  $\Omega\left(\log \log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$  queries are necessary.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms

**Keywords and phrases** Support-size estimation, Distribution testing, Conditional sampling, Lower bound

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.33

**Related Version** *Full Version:* <https://arxiv.org/abs/2211.11967>

**Funding** *Diptarka Chakraborty:* Supported in part by an MoE AcRF Tier 2 grant (MOE-T2EP20221-0009).

*Gunjan Kumar:* Supported in part by National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFAI1-2019-0004].

*Kuldeep S. Meel:* Supported in part by National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 grant MOE-T2EP20121-0011, and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114].

**Acknowledgements** The authors would like to thank anonymous reviewers for their useful suggestions and comments on an earlier version of this paper.



© Diptarka Chakraborty, Gunjan Kumar, and Kuldeep S. Meel;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 33; pp. 33:1–33:13

Leibniz International Proceedings in Informatics



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

## 1 Introduction

We consider the problem of estimating the support size of a distribution  $D$  over a domain  $\Omega$  (of size  $n$ ), which is defined as follows:

$$\text{SUPP}(D) := \{x \mid D(x) > 0\}.$$

We are interested in  $(\epsilon, \delta)$ -approximation<sup>1</sup> of the size of  $\text{SUPP}(D)$  (i.e.,  $|\text{SUPP}(D)|$ ). For simplicity in exposition, throughout this paper, we consider  $\delta$  to be a small constant (more specifically,  $1/3$ , which can be reduced to any arbitrary small constant.). The support size estimation is a fundamental problem in data science and finds a myriad of applications ranging from database management, biology, ecology, genetics, linguistics, neuroscience, and physics (see [30] and the references therein). Naturally, the distribution is not specified explicitly, and therefore, the complexity of the problem depends on the queries that one is allowed to the distribution. As such, the primary objective is to minimize the number of queries (aka *query complexity*).

Along with support size estimation, several other properties of distributions have attracted investigations over the past three decades (see [9]). As such, several query models have been considered by the research community. The simplest model SAMP only allows drawing independent and identically distributed samples from  $D$ . Valiant and Valiant [30] showed that to get an estimation up to an additive factor of  $\epsilon n$  (for any  $\epsilon > 0$ ),  $O(n/\epsilon^2 \log n)$  samples suffice, which was subsequently improved to  $O(\frac{n}{\log n} \log^2(1/\epsilon))$  by Wu and Yang [31]. Further, Wu and Yang proved that  $\Omega(\frac{n}{\log n} \log^2(1/\epsilon))$  samples are also necessary to get an estimate up to an additive error of  $\epsilon n$ . A natural extension to SAMP is called *probability-revealing sample* or PR-SAMP, due to Onak and Sun [27], wherein instead of just returning an independent sample  $x$  from  $D$  (as in SAMP), the oracle provides a pair  $(x, D(x))$  (i.e., a sample along with the probability assigned on it by  $D$ ). Onak and Sun showed that to estimate the support size up to an additive error of  $\epsilon n$ ,  $\Theta(1/\epsilon^2)$  samples are necessary and sufficient in the PR-SAMP model. The same upper bound for the PR-SAMP model was also implicit in the work by Canonne and Rubinfeld [8].

As we seek to explore more powerful models than PR-SAMP, a model of interest is DUAL [2, 22, 8] wherein we have access to two oracles: One is SAMP that provides a sample from  $D$ , and another is EVAL that given any  $x \in \Omega$ , outputs the value of  $D(x)$ . In the DUAL model, for any  $\epsilon_1, \epsilon_2 \in (0, 1]$ , distinguishing between whether the support size of  $D$  is at most  $\epsilon_1 n$  or at least  $\epsilon_2 n$  requires  $\Theta(1/(\epsilon_2 - \epsilon_1)^2)$  queries [8]. An extension of EVAL is CEVAL wherein for totally ordered domains, given  $x$ , CEVAL outputs  $\sum_{y \preceq x} D(x)$ . Similarly, CDUAL is an extension of DUAL where we have access to oracles SAMP and CEVAL. Caferov, Kaya, O'Donnell, and Say [7] showed that  $\Omega(\frac{1}{\epsilon^2})$  queries are needed in the CDUAL model to estimate the support size up to an additive factor of  $\epsilon n$ . However, to the best of our knowledge, no non-trivial result is known for the support size estimation problem with  $(1 + \epsilon)$  multiplicative error in the above models.

While SAMP, PR-SAMP, and DUAL are natural models, they are limiting in theory and practice as they fail to capture several scenarios wherein one is allowed more powerful access to the distribution under consideration. Accordingly, CFGM [13] and CRS [11]

<sup>1</sup> We want to estimate  $|\text{SUPP}(D)|$  by  $\hat{s}$  such that  $\frac{|\text{SUPP}(D)|}{(1+\epsilon)} \leq \hat{s} \leq (1+\epsilon)|\text{SUPP}(D)|$  with the success probability at least  $1 - \delta$ . This version is also referred to as  $(1 + \epsilon)$ -multiplicative factor estimation. Another interesting version to consider is the additive  $\epsilon n$ -estimation (where  $n$  denotes the size of the domain) which asks to output a  $\hat{s}$  such that  $|\text{SUPP}(D)| - \epsilon n \leq \hat{s} \leq |\text{SUPP}(D)| + \epsilon n$  with the success probability at least  $1 - \delta$ . Unless otherwise stated explicitly, we consider the multiplicative variant.

initiated the study of a more general sampling model COND, where we are allowed to draw samples conditioning on any arbitrary subsets of the domain  $\Omega$ . More specifically, the sampling oracle takes a subset  $S \subseteq \Omega$  chosen by the algorithm and returns an element  $x \in S$  with probability  $D(x)/D(S)$  if  $D(S) > 0$ . The models proposed by CFGM and CRS differ in their behavior for the case when  $D(S) = 0$ . The model proposed by CFGM [13] allows the oracle to return a uniformly random element from  $S$  when  $D(S) = 0$ . On the other hand, the COND model defined in CRS [11] assumes that the oracle (and hence the algorithm) returns “failure” and terminates if  $D(S) = 0$ <sup>2</sup>. Note that the COND model of CRS is more powerful than that of CFGM since, when  $D(S) = 0$ , in the former case, we get to know that  $D(S) = 0$ , whereas in the latter case, we get a uniformly random element of  $S$ .

The relative power of the COND model of CRS over that of CFGM is also exhibited in the context of support size estimation. Acharya, Canonne, and Kamath [1] designed an algorithm with query complexity  $\tilde{O}(\log \log n / \epsilon^3)$  in the COND model of CFGM to estimate the support size up to  $(1 + \epsilon)$  multiplicative factor under the assumption that the probability of each element is at least  $\Omega(1/n)$ . They also note that the assumption of a lower bound on the probability of each element is required for their techniques to work. It is worth highlighting that for the problem of support size estimation in the SAMP model, one needs a lower bound of  $\Omega(1/n)$  on the minimum probability assigned to any element in the support. Otherwise, one can assign a negligible probability to certain elements which will never be observed. In the COND model, such an assumption is not necessary. Indeed, a result of Falahatgar, Jafarpour, Orlitsky, Pichapati, and Suresh [18] implies that  $O(\log \log n + \frac{1}{\epsilon^2})$  queries are sufficient in the COND model of CRS for any arbitrary probability distribution, i.e., there is no requirement for the assumption of lower bound on the probability of each element. The key idea behind this result is that it is possible to decide whether the support size is  $> 2t$  or  $\leq t$  for any integer  $t \geq 0$  (for any arbitrary probability distributions) with high probability using only  $O(1)$  queries, even with weaker oracle access in which, given any  $S \subseteq \Omega$ , it can be determined whether  $S \cap \text{SUPP}(D) = \emptyset$  or not. On the lower bound side, we only know that at least  $\Omega(\sqrt{\log \log n})$  COND queries are necessary [13].

The two models were introduced in the context of uniformity testing, wherein the choice of how to handle the case of  $D(S) = 0$  did not make any significant differences. We would like to emphasize that CRS’s model is more powerful than that of CFGM, and thus any lower bound shown in the first one also provides the same in the latter one. Moreover, the model of CRS closely approximates the behavior of modern probabilistic programming systems [21]. Therefore, throughout this paper, we consider the COND model of CRS.

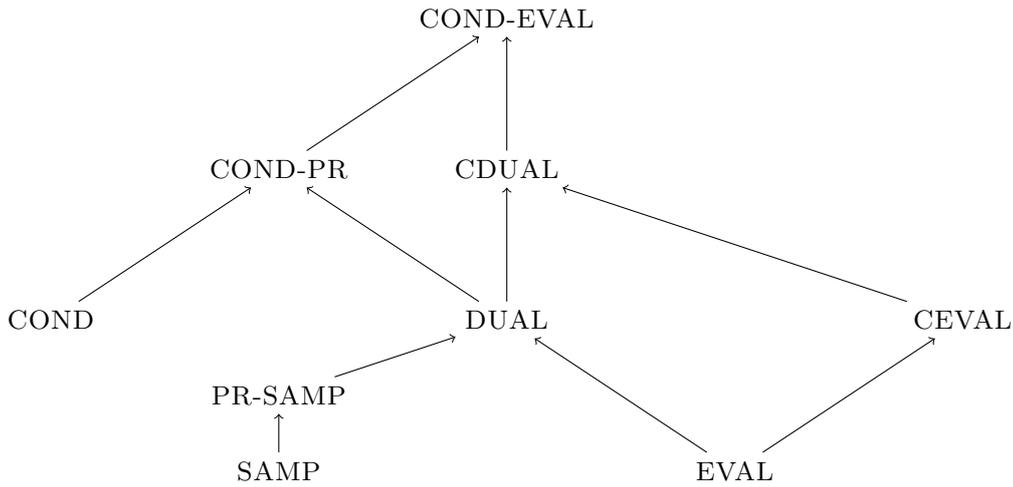
Since its introduction, the COND model has attained significant attention both in theory and practice. From a theoretical perspective, various other distribution testing problems have been studied under the COND model [17, 24, 26] and its variant like *subcube conditioning model* [3, 10, 15]. Apart from that, the COND model and its variants find real-world applications in the areas like formal methods and machine learning (e.g., [14, 25, 20]). Also, the modern probabilistic programming systems extend classical programs with the addition of sampling and *observe*, where the semantics of the observe match that of CRS’s COND model [21].

---

<sup>2</sup> Note that analyzing each step of the algorithm makes it possible to determine the set, conditioning on which caused the algorithm to output “failure” and terminate. The rest of the algorithm can then execute with the information that  $D(S) = 0$  for the above set  $S$ . Therefore, for the simplicity of exposition, we will assume that the COND model defined in CRS returns “failure” when  $D(S) = 0$ , but the algorithm does not terminate.

### 33:4 Support Size Estimation: The Power of Conditioning

It is worth remarking that the COND model is incomparable with PR-SAMP. Therefore, it is quite natural, both from a theory and practical perspective, to consider a sampling model that inherits power from both COND and PR-SAMP. We consider a model where we are allowed to condition on any arbitrary subset  $S \subseteq \Omega$ , and if  $D(S) > 0$ , we receive a sample  $x \in S$  with probability  $D(x)/D(S)$  (as in COND) along with the probability assigned on it by  $D$  (i.e.,  $D(x)$ ); “failure” otherwise. We refer to this model as *probability-revealing conditional sample*<sup>3</sup> or in short COND-PR. To the best of our knowledge, Golia, Juba, and Meel [20] were the first ones to initiate the study of the COND-PR model. Their work focused on the multiplicative estimation of entropy on the COND-PR model. Golia et al. were primarily motivated to investigate the COND-PR model upon the observation that the usage of the model counter and a sampler can simulate the COND-PR model wherein a circuit specifies the distribution. Also, implicit in their study is that the availability of model counter and samplers [16] allows one to simulate generalization of COND-PR model wherein for a given input  $D$  and  $S$  in addition to  $D(x)$  for a sampled item  $x \in S$ , the oracle also returns the value of  $D(x)/D(S)$  (the conditional probability of  $x$  given  $S$ ). We refer to this model as *conditional sampling evaluation model*<sup>4</sup>, or in short COND-EVAL. To the best of our knowledge, the COND-EVAL subsumes all the previously studied variants of the COND model (see Figure 1).



■ **Figure 1** Relative power of different models: An edge  $u \rightarrow v$  means the model  $v$  is more powerful than the model  $u$ .

To summarize, there has been a long line of research that has relied on the usage of the COND model and its variants, resulting in significant improvements in the query complexity for several problems in distribution testing. While there has been a multitude of techniques for obtaining upper bounds for the COND model and its variants, such has not been the case for lower bounds. In particular, the prior techniques developed in the context of support size estimation for COND model have primarily relied on the observation that an algorithm  $\mathcal{A}$  that makes  $q$  COND queries can be simulated by a decision tree of  $O(q2^{q^2})$  nodes. Accordingly, the foregoing observation allows one to obtain  $\Omega(\sqrt{\log \log n})$  lower bound for

<sup>3</sup> The name is motivated from the PR-SAMP model [27].

<sup>4</sup> The name is motivated from the standard *evaluation* model EVAL [28] where given any  $x \in \Omega$ , we get the value of the probability density function of  $D$  at  $x$ .

COND, which leaves open a major gap with respect to the upper bound of  $O(\log \log n + \frac{1}{\epsilon^2})$ . The situation is even direr when considering models that augment COND with powerful oracles such as EVAL since the approach based on a decision tree fails due to additional information supplied by oracles such as EVAL, and accordingly, no non-trivial lower bounds are known for models such as CDUAL, COND-EVAL, and the like. Therefore, there is a desperate need for new lower bound techniques to fully understand the power of the COND model and its natural extensions.

## 1.1 Our Contribution

One of our primary contributions is to provide a seemingly new approach to proving lower bounds in the COND model and its more powerful variants. Our approach is based on information theory and reductions to problems in communication complexity. We note that the communication complexity-based approaches to lower bounds have been explored in prior work; such approaches are only limited to weaker models such as the SAMP and the PAIRCOND models [6, 5]. While we demonstrate the application of our approach in the context of support size estimation for different variants of COND, we believe our approach is of general interest and can be applied to other distribution testing problems.

For ease of exposition, here we situate the discussion in the context of the most general model, COND-EVAL. One of the inherent difficulties in proving any non-trivial lower bound for the COND-EVAL model arises from the fact that the different sets for conditioning can overlap in an arbitrary manner (and unlike in PAIRCOND model, these sets are of arbitrary size) and further be chosen in an adaptive way. The adaptivity and arbitrary size of sets make it extremely difficult to upper bound the conditional entropy at any step of the algorithm. Furthermore, the power of revealing the probability mass (on any set) by a COND-EVAL query risks licking “a lot of” information which makes it even more challenging. The key departure from earlier work is the choice of an infinite family of distributions, so the range of outcomes of an algorithm is continuous and so cannot be encoded with any finite (or even infinite) length message. To this end, we rely on Fano’s inequality, a fundamental tool in information theory, to show lower bounds for statistical estimation. In order to apply Fano’s inequality, we need to upper bound the information gain at every step of the algorithm. Our approach proceeds by relying on a restricted model of conditioning where the queried sets are *laminar*, i.e., either they do not intersect or are subsets/supersets of each other. Accordingly, we first obtain lower bounds for the restricted model and lift to the lower bounds for the COND-EVAL model.

Our approach is compelling enough to provide non-trivial lower bounds in the most potent COND-EVAL model, for which no lower bound was known before. However, before providing the usefulness of the general framework, let us demonstrate how a special instantiation of our approach can be applied to obtain strong lower bounds in the context of support size estimation for the COND model and its (simpler) variants. Our first result bridges the long-standing gap between the upper bound of  $O(\log \log n + \frac{1}{\epsilon^2})$  and the lower bound of  $\Omega(\sqrt{\log \log n})$  in case of COND model. In particular, we obtain an  $\Omega(\log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)})$  lower bound on the query complexity in the COND-PR model, which in turn provides the same lower bound for the PR-SAMP, DUAL, and COND model.

► **Theorem 1.** *Every algorithm that, given COND-PR access to a distribution  $D$  on  $[n]$  and  $\epsilon \in (0, 1]$ , estimates the support size  $|\text{SUPP}(D)|$  within a multiplicative  $(1 + \epsilon)$ -factor with probability at least  $\frac{2}{3}$ , must make  $\Omega\left(\log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$  queries to the COND-PR oracle.*

## 33:6 Support Size Estimation: The Power of Conditioning

Recall the best-known upper bound for the support size estimation in COND is  $O(\log \log n + \frac{1}{\epsilon^2})$  and therefore, the above theorem achieves a near-matching lower bound in the context of COND and COND-PR. It is rather surprising that the combination of PR-SAMP and COND does not yield power in the context of the support size estimation. It is worth mentioning that we already know of a lower bound of  $\Omega(1/\epsilon^2)$  for the PR-SAMP and DUAL model due to [8]. Hence, our result along with [8] provides a lower bound of  $\Omega(\log \log n + \frac{1}{\epsilon^2})$  for these two models (i.e., we can get rid of the annoying  $\log(1/\epsilon)$  factor from the lower bound term of Theorem 1).

Our primary result is to establish the first non-trivial lower bound in the context of COND-EVAL, which in turn provides the first-known lower bound for many other previously studied models, such as CDUAL.

► **Theorem 2.** *Any algorithm that, given COND-EVAL access to a distribution  $D$  on  $[n]$  approximates the support size  $|\text{SUPP}(D)|$  within a multiplicative  $(1 + \epsilon)$ -factor with probability at least  $2/3$ , must make  $\Omega\left(\log \log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$  queries.*

Since COND-EVAL subsumes COND-PR, we have the upper bound of  $O(\log \log n + \frac{1}{\epsilon^2})$ , and the ensuing gap leaves open an interesting question. It is worth remarking that it is not hard to see (by extending the proof of Theorem 1) that the upper bound is nearly tight if one were to replace COND-EVAL with approximate-COND-EVAL wherein, for a given  $D$  and  $S$ , the oracle essentially provides an estimate of  $D(S)$  up to a small multiplicative error (see the discussion in the full version). We conjecture that the upper bound is tight for COND-EVAL.

► **Conjecture 3.** *Any algorithm that, given COND-EVAL access to a distribution  $D$  on  $[n]$  approximates the support size  $|\text{SUPP}(D)|$  within a multiplicative  $(1 + \epsilon)$ -factor with probability at least  $2/3$ , must make  $\Omega\left(\log \log n + \frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$  queries.*

The validity of the above conjecture would establish the significant power of the COND model in the context of support size estimation as COND-EVAL and COND-PR, despite being augmented with powerful oracles in addition to conditioning, do not yield better algorithms.

## 2 Preliminaries

### Notations

We use the notation  $[n]$  to denote the set of integers  $\{1, 2, \dots, n\}$ . For any probability distribution  $D$  defined over  $[n]$ , for any  $i \in [n]$ , let  $D(i)$  denote the probability of choosing  $i$  when sampling according to  $D$ . For any subset  $S \subseteq [n]$ , we use  $D(S)$  to denote the probability mass assigned on  $S$  by the distribution  $D$ , i.e.,  $D(S) := \sum_{i \in S} D(i)$ .

### Different access models

Let  $D$  be a distribution over  $[n]$ . Below we formally define the query models that we consider in this paper.

► **Definition 4 (COND Query Model).** *A conditional (in short, COND) oracle for  $D$  takes as input a set  $S \subseteq [n]$ , and if  $D(S) > 0$ , returns an element  $j \in S$  with probability  $D(j)/D(S)$ . If  $D(S) = 0$ , then the oracle returns “failure”.*

► **Definition 5** (COND-PR Query Model). A probability-revealing conditional sampling (in short, *COND-PR*) oracle for  $D$  takes as input a set  $S \subseteq [n]$ , and if  $D(S) > 0$ , returns a pair  $(j, D(j))$  (where  $j \in S$ ) with probability  $D(j)/D(S)$ . If  $D(S) = 0$ , then the oracle returns “failure”.

► **Definition 6** (COND-EVAL Query Model). A conditional evaluation (in short, *COND-EVAL*) oracle for  $D$  takes as input a set  $S \subseteq [n]$ , and if  $D(S) > 0$ , returns a tuple  $(j, D(j), D(j)/D(S))$  (where  $j \in S$ ) with probability  $D(j)/D(S)$ . If  $D(S) = 0$ , then the oracle returns “failure”.

► **Definition 7** (SET-EVAL Query Model). A set evaluation (in short, *SET-EVAL*) oracle for  $D$  takes as input a set  $S \subseteq [n]$ , and returns the value  $D(S)$ .

It is straightforward to observe that the COND-EVAL is at least as powerful as the COND-PR oracle which in turn is at least as powerful as the COND oracle. Further, the COND-EVAL oracle is at least as powerful as the SET-EVAL oracle.<sup>5</sup>

### Shannon entropy and source coding theorem

The *entropy* of a discrete random variable  $X$  taking values in  $\mathcal{X}$  is defined as  $H(X) := \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)}$  where  $p(x) = \Pr[X = x]$ .

The seminal work of Shannon [29] establishes a connection between the entropy and the expected length of an optimal code that encodes a random variable.

► **Theorem 8** (Shannon's Source Coding Theorem [29]). Let  $X$  be a discrete random variable over domain  $\mathcal{X}$ . Then for every uniquely decodable code  $C : \mathcal{X} \rightarrow \{0, 1\}^*$ ,  $\mathbb{E}(|C(X)|) \geq H(X)$ . Moreover, there exists a uniquely decodable code  $C : \mathcal{X} \rightarrow \{0, 1\}^*$  such that  $\mathbb{E}(|C(X)|) \leq H(X) + 1$ .

## 3 Technical Overview

### Lower bound for COND-PR

We start with deriving the lower bound of Theorem 1. It consists of two parts – an  $\Omega(\log \log n)$  lower bound for a multiplicative  $4/3$ -factor estimation algorithm and an  $\Omega(\frac{1}{\epsilon^2 \log \frac{1}{\epsilon}})$  lower bound for an additive  $\epsilon n$ -factor algorithm. We first show an  $\Omega(\log \log n)$  lower bound for a multiplicative  $4/3$ -factor algorithm. For that purpose, we consider an *integer-guessing* game between Alice and Bob, where Alice uniformly at random chooses an integer  $x \in [\log n]$ . Then sends a message (binary string) to Bob. Upon receiving the message, Bob's task is to guess  $x$  correctly (with high probability). Since the entropy of the chosen integer is  $\log \log n$ , by Shannon's source coding theorem, the length of the message, on average, must be  $\Omega(\log \log n)$ . We show that if there exists an algorithm  $T$  that makes  $t$  COND-PR queries, then it suffices for Alice to send a message of length  $O(t)$ , and hence  $t = \Omega(\log \log n)$ .

To show the same, for each  $x \in [\log n]$ , Alice considers a distribution  $D_x$  with support  $[2^x]$ . The probability  $D_x(j)$  of an element  $j \in [2^x]$  in  $D_x$  decreases exponentially as  $j$  increases. Alice runs the algorithm  $T$  on the distribution  $D_x$  and would like to send an encoding of the run (i.e., the sampled element along with its probability for each step). Using a trivial

<sup>5</sup> Since on input  $S$ , the COND-EVAL oracle returns a tuple  $(j, D(j), D(j)/D(S))$  where  $j \in S$ , one can compute the value of  $D(S)$  whenever  $D(S) > 0$ ; otherwise (when  $D(S) = 0$ ) the COND-EVAL oracle returns “failure”, from which one can infer that  $D(S) = 0$ .

encoding, even to send a sampled element, requires  $\Theta(\log n)$  bits, which is already more than the Shannon entropy and thus would not give any lower bound. So Alice needs to use a slightly clever encoding. Roughly speaking, since the probabilities are exponentially decreasing, the conditional sampling from any set  $S \subseteq [n]$  returns an element from the first “few” smallest elements of  $S$  (with high probability). Thus even though  $|S|$  can be large, the sampled element (at each step) can be specified by only constantly many bits. Alice sends this encoding of the sampled element (along with its probability value which can again be encoded with constantly many bits due to the construction of  $D_x$ 's) at each of  $t$  steps to Bob. Hence Bob knows the complete run and thus can determine the index  $x$  using the algorithm  $T$ . We provide detailed proof in the full version.

Next, we turn our attention to showing the dependency of  $\epsilon$  in the lower bound. We show an  $\Omega\left(\frac{1}{\epsilon^2 \log \frac{1}{\epsilon}}\right)$  lower bound for an additive  $\epsilon n$ -factor algorithm. Since a multiplicative  $(1 + \epsilon)$ -factor algorithm also provides an additive  $\epsilon n$ -factor estimation, the above lower bound also works for  $(1 + \epsilon)$ -factor algorithms. We prove this bound by showing a reduction from a well-studied communication complexity problem, namely the *Gap-Hamming distance* problem, and then applying the known lower bound for the Gap-Hamming distance [12]. The proof argument (and the dependence on  $\epsilon$ ) also holds true for COND-EVAL model.

### Lower bound for COND-EVAL

The approach used to get an  $\Omega(\log \log n)$  lower bound in the COND-PR model cannot be used to show the same for the COND-EVAL model. One of the powers of COND-EVAL model (over the COND-PR) comes from its ability to compute  $D(S) := \sum_{j \in S} D(j)$  for any set  $S \subseteq [n]$ .

Recall the hard instance  $D_x$ 's used in the  $\Omega(\log \log n)$  lower bound proof for the COND-PR model. Let  $X^* := \{2^x \mid x \in [\log n]\}$ . It is easy to verify that  $D_x(X^*) \neq D_{x'}(X^*)$  for any  $x \neq x' \in [\log n]$ . So, the value of  $x$  (and hence the support of  $D_x$ ) can be determined using only one COND-EVAL query with the set  $X^*$ . Thus our lower bound argument fails in this model. For the sake of intuition about how we overcome the above issue, we want to point out that the above argument does not fail if we have an approximate COND-EVAL query instead of COND-EVAL query, i.e., if the oracle gives the estimate of  $D(S)$  up to a small additive error say  $\frac{3}{2^{n^{0.1}}}$ . This is because for  $x, x' \in [\frac{\log n}{10}, \log n]$ , we have  $|D_x(X^*) - D_{x'}(X^*)| \leq \frac{2}{2^{n^{0.1}}}$ . Hence, given that  $x, x' \in [\frac{\log n}{10}, \log n]$ , we can not distinguish between  $x$  and  $x'$  as the estimate could be the same for both  $x$  and  $x'$ .

To mitigate the above issue (for COND-EVAL), we construct a new set of hard distributions. Our objective is that for any set  $X^*$ , if the value of  $D_x(X^*)$  is in  $(0, 1)$ , then value of  $D_x(X^*)$  should not give information about  $x$ . One plausible approach could be to replace each distribution  $D_x$  with a finite set of distributions such that for any set  $S \subseteq [n]$ , there are many distributions in the instance with the same value of  $D(S)$ . Unfortunately, we do not know how to get such a set of distributions preserving other useful properties needed for our proof. Our key high-level idea is to replace the distribution  $D_x$  (for each  $x \in [\log n]$ ) with a distribution over an infinite number of distributions. This way, the value of  $D_x(X^*)$  cannot be used to determine the value of chosen  $x$  (as there can be infinite values of  $x$  having the same value  $D(X^*)$ ). However, one immediate issue that arises is if our instance has an infinite domain (here distributions), then how do we even get a distribution over an infinite space? Further, like before, we still want the probabilities to exponentially decrease so that the sampled element is always among the first few smallest elements of the conditioning set  $S$ . Fortunately, in statistics and compositional data analysis, there has been a study of

probability distributions on the set of all (infinite) distributions. Based on requirements, our choice is the well-studied Dirichlet distribution that satisfies a strong independence property which is necessary for our analysis.

A Dirichlet distribution on support  $[K]$  with parameters  $\alpha_1, \dots, \alpha_K > 0$  has a probability density function given by  $f(p_1, \dots, p_K) = \frac{\prod_{i \in [K]} p_i^{\alpha_i - 1}}{B(\alpha_1, \dots, \alpha_K)}$  where  $\{p_i\}_{i \in [K]}$  belongs to the standard  $K-1$  simplex, i.e.,  $\sum_{i \in [K]} p_i = 1$  and  $p_i \geq 0$  for all  $i \in [K]$  and  $B(\alpha_1, \dots, \alpha_K)$  is a normalizing constant. When  $\alpha_1 = \dots = \alpha_K = 1$ , the Dirichlet distribution is just the uniform distribution on  $K-1$  simplex. The higher the value of parameter  $\alpha_i$ , the higher the (expected) value of  $p_i$ . Since we want the probability value to be exponentially decreasing, we set the values of  $\alpha_1, \dots, \alpha_n$  exponentially decreasing. Then for each index  $x$  chosen uniformly at random from  $[\log n]$ , we sample a distribution  $D_x$  with support  $[2^x]$  from the Dirichlet distribution with parameters  $\alpha_1, \dots, \alpha_{2^x}$ . By the standard Yao's principle, it suffices to show a high error probability of any deterministic algorithm that correctly estimates the support size (and hence determines the index  $x$ ) of the distribution sampled as above. Note that the entropy of the index is still  $\log \log n$ , but the previous communication framework (between Alice and Bob) will not be useful here. This is because the range of the outcomes of the algorithm (the actual and the conditional probabilities) is continuous, and so cannot be encoded with any finite (or even infinite) length message. Instead, we apply Fano's inequality, a tool from information theory.

Roughly speaking, we show that the information gain (about the index) by the query's outcome at every step is  $O(1)$ . Since the initial entropy of the index is  $\log \log n$ , at least  $\log \log n$  steps of the algorithm are needed. The main technical challenge is to upper bound the information gain at every step. It is particularly challenging as it requires calculating the explicit density function (for the outcome) corresponding to each index. These density functions are conditioned on the previous outcomes and thus change at every step. Further, the set queried by the algorithms can be adaptive, which makes our task even more difficult. To ease our analysis, we first assume that the queried sets by the algorithm are *laminar*, i.e., either they do not intersect or are subsets/supersets of each other. Our  $\Omega(\log \log n)$  lower bound holds for COND-EVAL model for all the algorithms satisfying this laminar condition. It is not hard to observe that any algorithm that makes  $t$  general queries can be simulated by an algorithm that queries the laminar family of sets and makes at most  $2^t$  queries. This observation gives us  $\Omega(\log \log \log n)$  lower bound for the general case. We believe that  $\Omega(\log \log n)$  is the correct lower bound for the general case, but (perhaps it is an artifact of our analysis that) the laminar structure is necessary for applying the independence properties of Dirichlet distribution which leads to only an  $\Omega(\log \log \log n)$  lower bound. For the sake of simplicity, we first prove the lower bound for a weaker model called, SET-EVAL, and then extend to the general COND-EVAL model. We refer to the oracle that, given any  $S \subseteq [n]$ , just outputs the value of  $D(S)$ , as SET-EVAL oracle. Since using a COND-EVAL query, we can simulate a SET-EVAL query, the COND-EVAL model is at least as powerful as SET-EVAL. We now describe the proof of the lower bound for the SET-EVAL model in more detail.

For an index  $x$  chosen uniformly at random from  $[\log n]$ , we sample a distribution  $D_x$  with support  $[2^x]$  from the Dirichlet distribution with parameters  $\alpha_1, \dots, \alpha_{2^x}$  ( $\alpha_j = \frac{1}{2^j}$  for all  $j \in [x]$ ). By the standard Yao's principle, it suffices for us to show a high error probability of any deterministic algorithm that correctly estimates the support size (and hence determines the index  $x$ ) of the distribution sampled as above. Let  $T$  be any such deterministic algorithm that queries a laminar family of sets. We need to show that  $T$  must make  $\Omega(\log \log n)$  queries. The main technical ingredient in the proof is to show that the information gain (about

### 33:10 Support Size Estimation: The Power of Conditioning

the index  $x$ ) by the outcome of any query of the algorithm is  $O(1)$ . This implies that the total information gain is  $O(t)$ , where  $t$  is the number of queries. Then Fano's inequality immediately implies that  $t \geq \Omega(\log \log n)$ . Let the  $i$ -th query ( $i \in [t]$ ) be denoted by set  $A_i$ , and the outcome of the  $i$ -th query (sum of probabilities of elements in the set  $A_i$ ) be denoted by  $Z_i$ . The information gain (about the index  $x$ ) by the  $i$ -th query (for any  $i \in [t]$ ) is the conditional mutual information  $I(X; Z_i | Z^{i-1})$  where  $Z_i$  is the random variable denoting the outcome of  $i$ -th query,  $Z^{i-1} = (Z_1, \dots, Z_{i-1})$  is the random variable denoting the vector of previous outcomes and  $X$  is the random variable denoting the uniformly chosen index  $x$  from  $[\log n]$ . By definition, the conditional mutual information  $I(A; B | C)$  for random variables  $A, B, C$  is equal to the expectation (over  $C$ ) of the KL divergence between the joint distribution  $Q_{(A,B)|C}$  and the product distribution  $Q_{A|C} \times Q_{B|C}$ . The technical difficulty is to upper bound the conditional mutual information  $I(X; Z_i | Z^{i-1})$  by  $O(1)$ . This is particularly challenging since the joint and the product distributions are not explicitly given, and queries are adaptive. To overcome this difficulty, at any step  $i \in [t]$  (after  $(i-1)$ -th query), we first partition the  $[\log n]$  into four groups denoted by  $L_0^i, L_1^i, L_2^i, L_3^i$  such that for the first three groups, the outcome of the algorithm (i.e.,  $Z_i$ ) is (deterministically) determined by the previous outcomes of the algorithm (i.e.,  $Z^{i-1} = (Z_1, \dots, Z^{i-1})$ ) whereas for any  $x$  in the fourth group  $L_3^i$ , the outcome is not fixed (given previous outcomes) but comes from a distribution (which we show to be also Dirichlet). Let this distribution be denoted by  $Q_x$  for  $x \in L_3^i$ . The information gain by the  $i$ -th query is  $\log 3$  (because there are three groups for which outcome is deterministically determined) plus the information gain corresponding to the last group  $L_3^i$ . This term can be upper bounded by the maximum KL divergence between distributions  $Q_x$  and  $Q_{x'}$  for any  $x, x' \in L_3^i$ . Thus our goal is to show that KL divergence between  $Q_x$  and  $Q_{x'}$  for any  $x, x' \in L_3^i$  is  $O(1)$ . Using the independence property of Dirichlet distributions and the laminar structure of query sets, we show that the KL divergence between the distributions  $Q_x$  and  $Q_{x'}$  is equal to the KL divergence between two beta distributions with different parameters (beta distributions are a special case of Dirichlet distributions). The explicit formula for KL divergence between two beta distributions is well-known (e.g., see [23]), and we use this formula to upper bound the KL divergence by  $O(1)$ .

► **Remark 9.** Technically, the above lower bound for SET-EVAL (and COND-EVAL) does not hold if it is promised that the probabilities in the given distribution are rational numbers. This is because, in the lower bound instances above, the probability of an element can be any arbitrary real number in  $(0, 1)$ . However, we can use the following standard fact to show that the lower bound holds even with the rational probabilities.

A Polya urn is an urn containing  $\alpha_i$  balls of color  $i$ , for each  $i \in [K]$ . The urn evolves at each discrete time step – a ball is sampled uniformly at random. The ball's color is observed, and two balls of the observed color are returned to the urn. Let  $X_{i,m}$  be the number of balls of color  $i$  (for each  $i \in [K]$ ) added after  $m$  time steps. Clearly  $D_m = (\frac{X_{1,m}}{m}, \dots, \frac{X_{K,m}}{m})$  is a probability distribution over  $[K]$ . It can be shown that the distributions  $D_1, \dots, D_m$  converges to a Dirichlet distribution with parameters  $\alpha_1, \dots, \alpha_K$  when  $m$  tends to  $\infty$  [4].

Instead of using Dirichlet distribution in our lower bound proof, we can use the distribution  $D_m$  for sufficiently large  $m$ . Since for any  $m$ , the probabilities in  $D_m$  are rational numbers, we can get the lower bound even when the probabilities are rationals.

#### The power of COND-EVAL

We further demonstrate the power of the COND-EVAL model by showing an algorithm with constant query complexity for a number of distribution testing problems for which there are strong lower bounds known for the COND-PR and COND model. Our first

example is the well-studied *Equivalence testing* problem. Here, given two distributions  $D$  and  $D'$ , the goal is to accept if  $D = D'$  and reject if their *total variation distance*  $\|D - D'\|_{TV} = \sum_{i \in [n]} |D(i) - D'(i)| > \epsilon$  (both with high probability). It is known that  $\Omega(\sqrt{\log \log n})$  queries are necessary in the COND model [1]. On the other hand,  $\Omega(1/\epsilon)$  queries are required in the COND-PR model. (Consider a uniform distribution  $U$  on  $[n]$ . Now randomly choose  $i, j \in [n]$ . We modify  $U$  to construct another distribution  $U'$  by setting  $U'(i) = 2/n$ ,  $U'(j) = 0$ , and no changes in the probability mass of other elements. Note that  $\epsilon = \|U - U'\|_{TV} = 2/n$ . It is easy to see that  $\Omega(n) = \Omega(\frac{1}{\epsilon})$  queries are required to distinguish  $U$  and  $U'$  in the COND-PR model.) We show that Equivalence testing can be done in just two COND-EVAL queries. The above upper bound result extends to another unrelated problem for the COND-EVAL model – the problem of testing whether the given distribution is *m-grained*, i.e., the probability of each element is an integer multiple of  $1/m$ . Finally, we show that the multiplicative  $(1 + \epsilon)$ -approximation of square of the  $L_2$  norm  $(\sum_{j \in [n]} D(j)^2)$  of a distribution  $D$  can be computed using  $O(\frac{1}{\epsilon^2})$  queries of COND-EVAL. To the best of our knowledge, this problem has been studied previously only in the SAMP model [19], wherein it was shown that  $\Omega(\frac{\sqrt{n}}{\epsilon^2})$  queries are required.

### The power of bounded-set conditioning

We further study the support size estimation problem when we allow SAMP oracle access and conditioning on sets of size at most  $k$ . We show a lower bound of  $\Omega(n/k)$  and an upper bound of  $O(\frac{n \log \log n}{k})$  for constant factor approximation in this model. The upper bound holds for the COND oracle model, while our lower bound holds for the stronger COND-EVAL oracle model.

Both the upper and lower bounds are not difficult to establish. Falahatgar et al. [18] showed that  $O(\log \log n)$  queries are sufficient (with no restriction on the size of the set for conditioning) to get a constant approximation for oracle access which, given any  $S \subseteq [n]$ , returns whether  $S \cap \text{SUPP}(D) = \emptyset$  or not. Oracle access to a set of size  $s$  can be simulated by  $s/k$  oracle access when conditioning on at most  $k$ -sized sets is allowed. This gives an upper bound of  $O(\frac{n \log \log n}{k})$ . Interestingly, the hard instance for the bounded-set conditioning to estimate the support size is when the support size is constant. We refer the readers to the full version for all the detailed proofs.

## 4 Conclusion

We investigate the power of conditioning for estimating the support size up to a multiplicative  $(1 + \epsilon)$ -factor. To date, there is a gap between the upper bound of  $O(\log \log n + 1/\epsilon^2)$  and the lower bound of  $\Omega(\sqrt{\log \log n})$  in the standard COND model. In this paper, we close this gap by providing a lower bound of  $\Omega(\log \log n + \frac{1}{\epsilon^2 \log \frac{1}{\epsilon}})$ . We actually show the lower bound in even a more powerful model, namely COND-PR, where in addition to the conditioning, one is also allowed to get the actual probability of the sampled elements (i.e., a combination of COND and PR-SAMP). In the dependency of  $\epsilon$ , there is a small gap of  $\log(1/\epsilon)$  factor, and we want to leave the problem of removing this factor from the lower bound term as an open problem.

It is quite surprising that the combination of COND and PR-SAMP does not yield more power compared to only the COND model in the context of the support size estimation. We thus continue our investigation by appending the algorithms with an even more powerful oracle that could also get the conditional probabilities of the sampled elements (not just the actual probabilities). We call this model COND-EVAL. This model turns out to be

more powerful in the context of several other important distribution testing problems. For the support size estimation, we show a lower bound of  $\Omega(\log \log \log n + \frac{1}{\epsilon^2 \log \frac{1}{\epsilon}})$  in this COND-EVAL model. On the technical side, this paper introduces many new ideas, such as using continuous distribution (Dirichlet distribution) for constructing hard instances and applying information theory and communication complexity tools to conditional sampling models. We hope that such techniques could be useful for showing non-trivial lower bounds for other distribution testing problems as well.

For the support size estimation problem in the COND-EVAL model, currently, we only know of an  $O(\log \log n)$  upper bound, whereas we could only show a lower bound of  $\Omega(\log \log \log n)$ . We would like to pose the problem of closing this gap as an interesting open problem. Another interesting open problem is to determine if an upper bound of  $o(\log \log n)$  is possible in the COND-PR and COND-EVAL models, assuming a certain lower bound on the probability mass of each element in the support size (e.g., the probability of any element is either 0 or at least  $1/n$ ).

---

## References

- 1 Jayadev Acharya, Clément L Canonne, and Gautam Kamath. A chasm between identity and equivalence testing with conditional queries. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.
- 2 Tugkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating the entropy. *SIAM Journal on Computing*, 35(1):132–150, 2005.
- 3 Rishiraj Bhattacharyya and Sourav Chakraborty. Property testing of joint distributions using conditional samples. *ACM Transactions on Computation Theory (TOCT)*, 10(4):1–20, 2018.
- 4 David Blackwell and James B MacQueen. Ferguson distributions via pólya urn schemes. *The annals of statistics*, 1(2):353–355, 1973.
- 5 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *computational complexity*, 21(2):311–358, 2012.
- 6 Eric Blais, Clément L Canonne, and Tom Gur. Distribution testing lower bounds via reductions from communication complexity. *ACM Transactions on Computation Theory (TOCT)*, 11(2):1–37, 2019.
- 7 Cafer Caferov, Barış Kaya, Ryan O’Donnell, and AC Say. Optimal bounds for estimating entropy with pmf queries. In *International Symposium on Mathematical Foundations of Computer Science*, pages 187–198. Springer, 2015.
- 8 Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2014.
- 9 Clément L Canonne. A survey on distribution testing: Your data is big. but is it blue? *Theory of Computing*, pages 1–100, 2020.
- 10 Clément L Canonne, Xi Chen, Gautam Kamath, Amit Levi, and Erik Waingarten. Random restrictions of high dimensional distributions and uniformity testing with subcube conditioning. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 321–336. SIAM, 2021.
- 11 Clément L Canonne, Dana Ron, and Rocco A Servedio. Testing probability distributions using conditional samples. *SIAM Journal on Computing*, 44(3):540–616, 2015.
- 12 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.
- 13 Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. *SIAM Journal on Computing*, 45(4):1261–1296, 2016.

- 14 Sourav Chakraborty and Kuldeep S Meel. On testing of uniform samplers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33(01), pages 7777–7784, 2019.
- 15 Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. Learning and testing junta distributions with sub cube conditioning. In *Conference on Learning Theory*, pages 1060–1113. PMLR, 2021.
- 16 Remi Delannoy and Kuldeep S Meel. On almost-uniform generation of sat solutions: The power of 3-wise independent hashing. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2022.
- 17 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Faster algorithms for testing under conditional sampling. In *Conference on Learning Theory*, pages 607–636. PMLR, 2015.
- 18 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Estimating the number of defectives with group testing. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1376–1380. IEEE, 2016.
- 19 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011.
- 20 Priyanka Golia, Brendan Juba, and Kuldeep S. Meel. Efficient entropy estimation with applications to quantitative information flow. In *International Conference on Computer-Aided Verification (CAV)*, 2022.
- 21 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Future of Software Engineering Proceedings, FOSE 2014*, pages 167–181, New York, NY, USA, 2014. Association for Computing Machinery. doi: 10.1145/2593882.2593900.
- 22 Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Sublinear estimation of entropy and information distances. *ACM Transactions on Algorithms (TALG)*, 5(4):1–16, 2009.
- 23 Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions, volume 2*, volume 289. John wiley & sons, 1995.
- 24 Gautam Kamath and Christos Tzamos. Anaconda: A non-adaptive conditional sampling algorithm for distribution testing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 679–693. SIAM, 2019.
- 25 Kuldeep S Meel, Yash Pralhad Pote, and Sourav Chakraborty. On testing of samplers. *Advances in Neural Information Processing Systems*, 33:5753–5763, 2020.
- 26 Shyam Narayanan. On tolerant distribution testing in the conditional sampling model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 357–373. SIAM, 2021.
- 27 Krzysztof Onak and Xiaorui Sun. Probability-revealing samples. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018.
- 28 Ronitt Rubinfeld and Rocco A Servedio. Testing monotone high-dimensional distributions. *Random Structures & Algorithms*, 34(1):24–44, 2009.
- 29 C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.
- 30 Gregory Valiant and Paul Valiant. Estimating the unseen: an  $n/\log(n)$ -sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 685–694, 2011.
- 31 Yihong Wu and Pengkun Yang. Chebyshev polynomials, moment matching, and optimal estimation of the unseen. *The Annals of Statistics*, 47(2):857–883, 2019.



# Query Complexity of Search Problems

Arkadev Chattopadhyay ✉ 

Tata Institute of Fundamental Research, Mumbai, India

Yogesh Dahiya ✉ 

The Institute of Mathematical Sciences (a CI of Homi Bhabha National Institute), Chennai, India

Meena Mahajan ✉ 

The Institute of Mathematical Sciences (a CI of Homi Bhabha National Institute), Chennai, India

---

## Abstract

We relate various complexity measures like sensitivity, block sensitivity, certificate complexity for multi-output functions to the query complexities of such functions. Using these relations, we provide the following improvements upon the known relationship between pseudo-deterministic and deterministic query complexity for total search problems:

- We show that deterministic query complexity is at most the third power of its pseudo-deterministic query complexity. Previously, a fourth-power relation was shown by Goldreich, Goldwasser and Ron (ITCS'13).
- We improve the known separation between pseudo-deterministic and randomized decision tree size for total search problems in two ways: (1) we exhibit an  $\exp(\tilde{\Omega}(n^{1/4}))$  separation for the SEARCHCNF relation for random  $k$ -CNFs. This seems to be the first exponential lower bound on the pseudo-deterministic size complexity of SEARCHCNF associated with random  $k$ -CNFs. (2) we exhibit an  $\exp(\Omega(n))$  separation for the APPROXHAMWT relation. The previous best known separation for any relation was  $\exp(\Omega(n^{1/2}))$ .

We also separate pseudo-determinism from randomness in AND and (AND, OR) decision trees, and determinism from pseudo-determinism in PARITY decision trees. For a hypercube colouring problem, that was introduced by Goldwasser, Impagliazzo, Pitassi and Santhanam (CCC'21) to analyze the pseudo-deterministic complexity of a complete problem in TFNP<sup>dt</sup>, we prove that either the *monotone* block-sensitivity or the *anti-monotone* block sensitivity is  $\Omega(n^{1/3})$ ; Goldwasser et al. showed an  $\Omega(n^{1/2})$  bound for *general* block-sensitivity.

**2012 ACM Subject Classification** Theory of computation → Oracles and decision trees; Theory of computation → Models of computation

**Keywords and phrases** Decision trees, Search problems, Pseudo-determinism, Randomness

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.34

**Related Version** *Full Version:* <https://eccc.weizmann.ac.il/report/2023/039/> [3]

## 1 Introduction

The question of whether randomness adds computational power over determinism, and if so, how much, has been a question of great interest that is still not completely understood. Naturally, the answer depends on the computational model under consideration, but it also depends on the type of problems one hopes to solve. One may wish to compute some function of the input, a special case being decision problems where the function has just two possible values. There are also search problems, where for some fixed relation  $R \subseteq X \times Y$  and an input  $x \in X$ , one wishes to find a  $y \in Y$  that is related to  $x$ ; i.e.  $(x, y) \in R$ . If every  $x \in X$  has at least one such  $y$ , we have a total search problem defined by  $R$ , the  $R$ -search problem. In the context of (total) search problems, a nuanced usage of randomness led to the beautiful notion of pseudo-determinism; see [7]. A function  $f$  solves the  $R$ -search problem if for every  $x$ ,  $(x, f(x)) \in R$ . A randomized algorithm which computes such an  $f$



© Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 34; pp. 34:1–34:15

Leibniz International Proceedings in Informatics



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

with high probability is said to be a pseudo-deterministic algorithm solving the  $R$ -search problem. Thus a pseudo-deterministic algorithm uses randomness to solve a search problem and almost always provides a canonical solution per input.

The original papers introducing pseudo-determinism examined both polynomial-time algorithms and sublinear-time algorithms; in the latter case, the computational resource measure is query complexity. In [8], a maximal separation was established between pseudo-deterministic and randomized query algorithms;  $\Omega(n)$  vs  $O(1)$ . Very recently, in [9], this separation was revisited. The separating problems in [8] do not lie in the query-complexity analogue of NP (nondeterministic polylog query complexity, or polylog query complexity to deterministically verify a solution,  $\text{TFNP}^{dt}$ ). This is a very natural class of search problems, and in [9], an almost-maximal separation between randomized and pseudo-deterministic search is established for a problem in this class. The problem in question is  $\text{SEARCHCNF}$ : given an assignment to the variables of a highly unsatisfiable  $k$ -CNF formula, to search for a falsified clause; this problem is very easy for randomized search ( $O(1)$  queries), and solutions are easily verifiable. Theorem 7 of [9] establishes that for unsatisfiable  $k$ -CNF formulas on  $n$  variables with sufficiently strong expansion in the clause-variable incidence graph (in particular, for most random  $k$ -CNF formulas), the corresponding search problem has pseudo-deterministic complexity  $\Omega(\sqrt{n})$ , even in the quantum query setting. In [9], the size measure of decision trees in the pseudo-deterministic setting was also studied. Lifting the query separation using a small gadget, a strong separation between randomized size and pseudo-deterministic size was obtained:  $\text{SEARCHCNF}$  problem on random  $k$ -CNFs lifted with 2-bit XOR has randomized size  $O(1)$  but require pseudo-deterministic size  $\exp(\Omega(\sqrt{n}))$ .

Taking this study further, Theorem 3 of [9] shows that the promise problem  $\text{PROMISEFIND1}$ , of finding a 1 in an  $n$ -bit string with Hamming weight at least  $n/2$ , is in a sense complete for the class of search problems that are in  $\text{TFNP}^{dt}$  and have efficient randomized query algorithms. By relating this search problem to a certain combinatorial problem concerning colourings of the hypercube, and by using the lower bound for  $\text{SEARCHCNF}$ , a lower bound of  $\Omega(\sqrt{n})$  on the pseudo-deterministic complexity of  $\text{PROMISEFIND1}$  is obtained (Theorem 14 and subsequent remark in [9]). The colouring problem on hypercubes states that any proper coloring of the hypercube contains a point with many 1s and with high block sensitivity. In [9], a point with block sensitivity  $\Omega(\sqrt{n})$  is proven to exist (Theorem 14), and a point with block sensitivity  $\Omega(n)$  is conjectured to exist (Conjecture 16).

### Our contributions

- (1) We improve upon the known relationship between pseudo-deterministic query complexity and deterministic query complexity for total search problems: We show that deterministic query complexity is at most the third power of its pseudo-deterministic query complexity. (Previously a fourth-power relation was shown in [8].)
- (2) We improve the known separations between pseudo-deterministic and randomized decision tree size in two ways: (1) an  $\exp(\tilde{\Omega}(n^{1/4}))$  separation for the  $\text{SEARCHCNF}$  relation for random  $k$ -CNFs (the  $\exp(\Omega(n^{1/2}))$  separation in [9] is only for the lifted formulas  $k$ -CNF composed with XOR), and (2) an  $\exp(\Omega(n))$  separation for the  $\text{APPROXHAMWT}$  relation (the previous best separation for any relation was  $\exp(\Omega(n^{1/2}))$ ).
- (3) We separate pseudo-deterministic and randomized query complexity in AND and (AND, OR) decision trees, and show that deterministic and pseudo-deterministic complexity are polynomially related in these models, upto polylog  $n$  factors. In the  $\text{PARITY}$  decision tree model, we observe that deterministic and pseudo-deterministic query complexities are well separated.

- (4) For the hypercube colouring problem posed in [9], we prove that either the monotone block-sensitivity or the anti-monotone block sensitivity is  $\Omega(n^{1/3})$ ; previously an  $\Omega(n^{1/2})$  bound was known but only for general block-sensitivity.

### Significance, context, and techniques

We now describe each of our contributions, with surrounding context, in more detail.

For Boolean functions, randomized and deterministic query complexities are known to be polynomially related by the classic result of Nisan [14]. Since deterministic query lower bounds are often easy to obtain using some kind of adversary argument, this provides a route to randomized query lower bounds for Boolean functions. For search problems, however, there is no such polynomial relation. Note that separating pseudo-determinism from randomness requires a lower bound against randomized query algorithms that provide canonical solutions. Such algorithms compute multi-output functions (following nomenclature from [9]) as opposed to Boolean functions. Thus what is required is randomized query lower bounds for multi-output functions. For such functions, too, lower bounds for deterministic querying are often relatively easy to obtain. And again, as for Boolean functions, deterministic and randomized query complexity for multi-output functions are known to be polynomially related; in [8] (Theorem 4.1(3)), the authors show that the deterministic query complexity is bounded above by the fourth power (as opposed to cubic power for Boolean functions) of the randomized complexity. They also show that it is bounded above by the cubic power times a factor that depends on the size of the search problem's range. We revisit these relations, and further tighten them to a cubic power relation. Thus for search problems, deterministic query complexity is bounded above by the cubic power of its pseudo-deterministic query complexity; Theorem 3.2. We show this by relating various complexity measures like sensitivity, block sensitivity, certificate complexity for multi-output functions to their query complexities; Theorem 3.1.

Using the recent result from [4] that derandomized the size measures for total Boolean functions, we establish a polynomial relationship between the log of pseudo-deterministic size and the log of deterministic size, ignoring polylog factors in the input dimension; Theorem 4.3. This gives us another way to separate randomized size from pseudo-deterministic size: any total search problem which is easy with randomization but difficult for deterministic search will lead to a separation between pseudo-deterministic size and randomized size; one such problem is SEARCHCNF on suitably expanding  $k$ -CNF formulas. In [9], it was shown that SEARCHCNF for such formulas *lifted* by small gadgets like XOR, has large pseudo-deterministic size complexity. There are known situations where the complexity of a formula and its lift by small gadgets widely vary in search problems. For instance, it was known that proving the unsatisfiability of formulas corresponding to Tseitin lifted by a small gadget should be hard in cutting planes proof system [6]. The popular belief was that such hardness extends to even unlifted Tseitin formulas. In a breakthrough work [5], this belief was proven false! However, we are able to obtain an  $\exp(\tilde{\Omega}(n^{1/4}))$  lower bound on the pseudo-deterministic size complexity for SEARCHCNF with unlifted random  $k$ -CNF formulas, in contrast to the bounds from [9]. As far as we know, this is the first exponential lower bound on the pseudo-deterministic size complexity of SEARCHCNF for random  $k$ -CNF formulas. Like Tseitin formulas, determining the complexity of random  $k$ -CNF formulas in various models remains an important theme of current research.

We also show, see Theorem 4.5, that any completion of the promise-problem APPROXMAJ by a total Boolean function, requires large randomized decision tree size. Observing that this promise-problem is “embedded” in the APPROXHAWWT search problem, we obtain an  $\exp(\Omega(n))$  separation between the pseudo-deterministic and randomized size complexity of APPROXHAWWT, in Theorem 4.6.

The more general query models we consider are those of AND (OR) decision trees, abbreviated as ADT's (ODT's), where each query is a conjunction of variables, (AND, OR) decision trees, where each query is either a conjunction or a disjunction of variables, and PARITY decision trees, where each query reports the parity of some subset of variables. These models obviously generalize decision trees, and are more powerful in the deterministic setting, appear naturally in contexts like combinatorial group testing and other contexts. More recently, they have been advocated by [11] as a meaningful intermediate model between query and communication complexity. For AND and (AND, OR) decision trees, we show that pseudo-determinism is still separated from randomness; Theorems 5.3 and 5.6. To show the former, we relate randomized query complexity for multi-output functions in this model to monotone block sensitivity. To show the latter, we note that a recently proved result from [4] relating depth in (AND, OR) trees and size in ordinary trees for Boolean functions, also holds for multi-output functions. Furthermore, using other results from [4] that derandomized the AND and (AND, OR) decision trees for total Boolean functions, we observe that pseudo-determinism and determinism are polynomially related in these settings, ignoring polylog $n$  factors; Theorems 5.4 and 5.7. For PARITY decision trees, in contrast, we observe that determinism is separated from pseudo-determinism; Theorem 5.8. There is no polynomial relation between these two complexity measures. In this setting, we do not know whether pseudo-determinism is separated from randomness.

Finally, we revisit the hypercube coloring problem from [9]. There, the existence of a point with large Hamming weight and block-sensitivity  $\Omega(\sqrt{n})$  is established, using the previously established lower bound for SEARCHCNF. We give a completely combinatorial and constructive argument to show that a point with large Hamming weight and block-sensitivity  $\Omega(n^{1/3})$  exists, Theorem 6.3. While we seemingly sacrifice stronger bounds in the quest for simplicity, our algorithm actually proves something that is stronger in a different way, and hence our result is perhaps incomparable with that of [9]. The difference is that we identify many sensitive blocks that are all 1's, or many sensitive blocks that are all 0's. In other words, we show that the monotone (or anti-monotone) block sensitivity is  $\Omega(n^{1/3})$ . Monotone block sensitivity was used recently, first by [12] and then by [4], to prove query complexity lower bounds for ADT's. In particular, our result implies that every function that solves PROMISEFIND1, requires large depth to be implemented by *either* randomized ADT's *or* by randomized ODT's. We believe that this could be strengthened to show that such solutions are always hard for randomized ADTs<sup>1</sup>. Proving such a result is an interesting open problem.

### Related work

For Boolean functions, the relations between many complexity measures and query complexity has been studied extensively in the literature. A consolidation of many known results appears in [2] as well as in [10].

### Organisation of the paper

In Section 3 we establish the relationships between various measures for multi-output functions, and establish the polynomial relation between pseudo-deterministic and deterministic query complexity for search problems. In Section 4 we establish relations between pseudo-deterministic size and deterministic size. Section 5 discusses the complexity of search problems

---

<sup>1</sup> Note that there are solutions that are easy for ODTs, even deterministically. For instance, a binary search can be implemented to find efficiently the first occurrence of a 1.

in AND, (AND, OR), and PARITY decision trees. Section 6 discusses the hypercube coloring problem. We refer the reader to the full version of our paper [3] for proofs omitted due to space constraints.

## 2 Preliminaries

We use standard notation and terminology in the paper, which we briefly explain here. For detailed definitions, we refer the reader to the full version of our paper [3]. For  $x \in \{0, 1\}^*$ , and  $b \in \{0, 1\}$ ,  $|x|$  is the length of  $x$  and  $|x|_b$  is the number of occurrences of  $b$  in  $x$ . We also write  $\text{wt}(x)$  for  $|x|_1$ . To sample uniformly from a set  $S$ , we write  $\sim_u S$ . For  $B \subseteq [n]$  and  $b \in \{0, 1\}$ ,  $b_B$  is the  $n$ -bit string that equals  $b$  at positions in  $B$  and  $1 - b$  elsewhere.

For a multi-output function  $f : \{0, 1\}^n \rightarrow [m]$  and input  $x \in \{0, 1\}^n$ , we use the following measures, which are natural extensions from the Boolean case, to capture different aspects of its complexity:  $s(f, x)$  counts the number of input bits at  $x$  that can be flipped to change the output of  $f(x)$ ;  $\text{bs}(f, x)$  is the maximum integer  $r$  for which there exist  $r$  disjoint sensitive blocks of  $f$  at  $x$  (where  $f$  is sensitive to block  $B$  on input  $x$  if  $f(x) \neq f(x \oplus 1_B)$ );  $C(f, x)$  is the minimum number of input bits required to uniquely identify the output of  $f(x)$ . Maximizing over all inputs gives  $s(f)$ ,  $\text{bs}(f)$ , and  $C(f)$ , the sensitivity, block sensitivity, and certificate complexity of  $f$ , respectively. We also use sensitivity and block sensitivity measures that only allow flipping either 0s or 1s. A set  $B \subseteq [n]$  is a *sensitive  $b$ -block* of  $f$  at input  $x$  if  $x_i = b$  for each  $i \in B$ , and  $f(x) \neq f(x \oplus 1_B)$ . The  *$b$ -block sensitivity* of  $f$  at  $x$ , denoted  $\text{bs}_b(f, x)$ , is the maximum integer  $r$  for which there exist  $r$  disjoint sensitive  $b$ -blocks of  $f$  at  $x$ . The  *$b$ -sensitivity* of  $f$  at  $x$ ,  $s_b(f, x)$ , is the number of sensitive  $b$ -bits of  $x$ . Maximizing over all inputs gives  $s_b(f)$ , and  $\text{bs}_b(f)$ , the  $b$ -sensitivity and  $b$ -block sensitivity of  $f$ , respectively. Note that  $s_0(f)$  and  $\text{bs}_0(f)$  are the same as the monotone sensitivity and monotone block sensitivity used in [12] to study AND-decision trees.

For a search problem  $\mathcal{S}$ , a (deterministic) decision tree  $T$  computing  $\mathcal{S}$  is a binary tree with internal nodes labelled by the variables and the leaves labelled by  $y \in \mathcal{Y}$ . It evaluates an unknown input  $x$  by traversing the tree based on variable queries. The label of the leaf reached must belong to  $\mathcal{S}(x)$ . The depth of a decision tree  $T$  is the length of the longest root-to-leaf path, and its size is the number of leaves. The deterministic query/size complexity of  $\mathcal{S}$ , denoted by  $D^{\text{dt}}(\mathcal{S})/D\text{Size}^{\text{dt}}(\mathcal{S})$ , is defined to be the minimum depth/size of a decision tree that computes  $\mathcal{S}$ .

A randomized query algorithm/decision tree  $\mathcal{A}$  is defined by a distribution  $\mathcal{D}_{\mathcal{A}}$  over deterministic decision trees. It evaluates an input  $x$  by randomly selecting a tree  $T$  from  $\mathcal{D}_{\mathcal{A}}$  and outputting the label of the leaf reached by  $T$  on  $x$ . The algorithm is considered to compute  $\mathcal{S}$  with error at most  $\epsilon$  if, for every  $x$ , the probability that  $A(x)$  belongs to  $\mathcal{S}(x)$  is at least  $1 - \epsilon$ . The depth/size complexity of the randomized decision tree is determined by the maximum depth/size among all decision trees in the distribution's support. The randomized query/size complexity of  $\mathcal{S}$  with error  $\epsilon$ , denoted by  $R_{\epsilon}^{\text{dt}}(\mathcal{S})/R\text{Size}_{\epsilon}^{\text{dt}}(\mathcal{S})$ , is the minimum depth/size of a randomized decision tree that computes  $\mathcal{S}$  with error  $\epsilon$ . For a probability distribution  $\mathcal{D}$  over the domain of  $\mathcal{S}$ , the  $(\mathcal{D}, \epsilon)$ -distributional query/size complexity of  $\mathcal{S}$ , denoted by  $D_{\mathcal{D}, \epsilon}^{\text{dt}}(\mathcal{S})/D\text{Size}_{\mathcal{D}, \epsilon}^{\text{dt}}(\mathcal{S})$ , is the minimum depth/size of a deterministic decision tree that gives a correct answer on  $1 - \epsilon$  fraction of inputs weighted by  $\mathcal{D}$ . When we drop  $\epsilon$  from the subscript of a randomized/distributional query measures, we assume  $\epsilon = 1/3$ .

A multi-output function  $f : \{0, 1\}^n \rightarrow [m]$  solves  $\mathcal{S}$ , denoted by  $f \in_s \mathcal{S}$ , if for every  $x \in \{0, 1\}^n$ ,  $(x, f(x)) \in S$ . A pseudo-deterministic query algorithm for a search problem  $\mathcal{S}$ , with error  $1/3$ , is a randomized decision tree that computes some multi-output function

$f$  that solves  $\mathcal{S}$  with error at most  $1/3$ . The pseudo-deterministic query complexity of  $\mathcal{S}$ , denoted by  $\text{psD}^{\text{dt}}(\mathcal{S})$ , is equal to  $\min_{f \in_s \mathcal{S}} R^{\text{dt}}(f)$ , and pseudo-deterministic size complexity of  $\mathcal{S}$ , denoted by  $\text{psDSize}^{\text{dt}}(\mathcal{S})$ , is equal to  $\min_{f \in_s \mathcal{S}} \text{RSize}^{\text{dt}}(f)$ . Note that randomized query algorithms for  $\mathcal{S}$  do not necessarily output a canonical value with high probability, only a value  $y$  such that  $(x, y) \in \mathcal{S}$  with high probability.

### Known results that we use

► **Proposition 2.1** ([15]). *For a search relation  $\mathcal{S}$ ,*  
 $R_\epsilon^{\text{dt}}(\mathcal{S}) = \max_{\mathcal{D}} D_{\mathcal{D}, \epsilon}^{\text{dt}}(\mathcal{S})$  and  $\text{RSize}_\epsilon^{\text{dt}}(\mathcal{S}) = \max_{\mathcal{D}} \text{DSize}_{\mathcal{D}, \epsilon}^{\text{dt}}(\mathcal{S})$ .

► **Proposition 2.2** ([14, 10, 2]). *For any Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,*

1.  $s(f) \leq bs(f) \leq C(f) \leq s(f)bs(f)$ .
2.  $s(f) \leq bs(f) \leq 3R_{1/3}^{\text{dt}}$ .
3.  $C(f) \leq D^{\text{dt}}(f) \leq C(f)^2$ .
4.  $D^{\text{dt}}(f) \leq C(f)bs(f)$ .
5.  $D^{\text{dt}}(f) = O((R^{\text{dt}}(f))^3)$ .

► **Proposition 2.3** (restated from [8]). *For a search relation  $\mathcal{S}$ ,*

1.  $D^{\text{dt}}(\mathcal{S}) \leq \left(\text{psD}^{\text{dt}}(\mathcal{S})\right)^4$ . [Restated from Theorem 4.1(3) in [8]]
2.  $D^{\text{dt}}(\mathcal{S}) \leq \left(\text{psD}^{\text{dt}}(\mathcal{S})\right)^3 \ell_{\mathcal{S}}(n)$ , where  $\ell_{\mathcal{S}}(n)$  is the number of bits required to represent the range of  $\mathcal{S}$ . [Restated from Theorem 4.1(3) in [8]]

► **Proposition 2.4.**

1. (Corollary 4.2 in [8]) *For the relation*  
 $\text{APPROXHAWWT} = \{(x, v) : |wt(x) - v| \leq n/10\}$ ,  
 $\text{psD}^{\text{dt}}(\text{APPROXHAWWT}) \in \Omega(n)$  and  $R^{\text{dt}}(\text{APPROXHAWWT}) = O(1)$ .
2. (Theorem 4 in [9]) *For the relation*  $\text{PROMISEFIND1} = \{(x, i) : wt(x) \geq |x|/2 \wedge x_i = 1\}$ ,  
 $\text{psD}^{\text{dt}}(\text{PROMISEFIND1}) \in \Omega(\sqrt{n})$  and  $R^{\text{dt}}(\text{PROMISEFIND1}) = O(1)$ .

► **Proposition 2.5.** *For  $F$  a random 3-CNF formula on  $n$  variables with  $m = \Theta(n)$  clauses sampled from  $\mathcal{F}_m^{3,n}$ , with probability  $1 - o(1)$ ,  $F$  is unsatisfiable and furthermore,*

1.  $R^{\text{dt}}(\text{SEARCHCNF}(F)) = O(1)$ .
2.  $D^{\text{dt}}(\text{SEARCHCNF}(F)) = \Omega(n)$ . (From [13, 1])
3.  $\text{DSize}^{\text{dt}}(\text{SEARCHCNF}(F)) = \exp(\Omega(n))$ . (From [1])

## 3 Relating measures for multi-output functions

We show the analogs of Proposition 2.2 for multi-output functions. The idea is to do the necessary modifications to the analogous results in the Boolean function case. For the proof, we refer the reader to the full version of our paper [3].

► **Theorem 3.1.** *For a function  $f : \{0, 1\}^n \rightarrow [m]$ , the following relations hold.*

1.  $C(f) \leq s(f)bs(f)$ .
2.  $s(f) \leq bs(f) \leq 3R_{1/3}^{\text{dt}}(f)$
3.  $C(f) \leq D^{\text{dt}}(f) \leq C(f)^2$ .
4.  $D^{\text{dt}}(f) \leq 2C(f)bs(f)$ .
5.  $D^{\text{dt}}(f) = O((R^{\text{dt}}(f))^3)$ .

Using the above, we can now improve the bounds from Proposition 2.3 for search problems. One  $\text{psD}^{\text{dt}}(\mathcal{S})$  factor from item 1 there can be removed, as also the  $\ell_{\mathcal{S}}(n)$  factor in item 2.

► **Theorem 3.2.** For any total search problem  $\mathcal{S}$ ,  $D^{\text{dt}}(\mathcal{S}) = O((\text{psD}^{\text{dt}}(\mathcal{S}))^3)$ .

**Proof.** For total search problem  $\mathcal{S}$ , let  $\tilde{f}$  be a function solving  $\mathcal{S}$ , with  $\text{psD}^{\text{dt}}(\mathcal{S}) = R^{\text{dt}}(\tilde{f})$ . Then, using Theorem 3.1(5), we obtain  $D^{\text{dt}}(\mathcal{S}) = \min_{f \in_s \mathcal{S}} D^{\text{dt}}(f) \leq D^{\text{dt}}(\tilde{f}) \leq O((R_{1/3}^{\text{dt}}(\tilde{f}))^3) = O(\text{psD}^{\text{dt}}(\mathcal{S})^3)$ . ◀

## 4 Pseudo-deterministic size vs deterministic size

In this section, we establish a polynomial relationship, up to polylog  $n$  factors, between the logarithm of pseudo-deterministic size and the logarithm of deterministic size for total search problems, Theorem 4.3. We also improve the separation between pseudo-deterministic and randomized size, Theorem 4.6.

Before showing these results, we first examine an argument for extending results on Boolean functions to multi-output functions. We note that a relationship between randomized and deterministic complexity in a query model for Boolean functions yields an almost identical relationship between pseudo-deterministic complexity and deterministic complexity for search problems. The result follows from a straightforward application of a binary search argument and also appears in the work of [8] for making a similar claim for the ordinary query model. We give a proof sketch, for details refer to the full version of our paper [3].

► **Proposition 4.1.** In a query model  $M$ , let  $D^M$  ( $\text{DSize}^M$ ),  $R^M$  ( $\text{RSize}^M$ ) and  $\text{psD}^M$  ( $\text{psDSize}^M$ ) denote the query complexity (size complexity, respectively) in the deterministic, randomized and pseudo-deterministic settings. Then,

1. If for some monotonic function  $q : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  and every total Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $D^M(f) \leq q(R^M(f), n)$ , then for any total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ ,  $D^M(\mathcal{S}) = O(q(\text{psD}^M(\mathcal{S}), n) \cdot \min(\log m, \text{psD}^M(\mathcal{S})))$ .
2. If for some monotonic function  $q : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{N}$  and every total Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,  $\log \text{DSize}^M(f) \leq q(\log \text{RSize}^M(f), n)$ , then for any total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ ,  $\log \text{DSize}^M(\mathcal{S}) = O(q(\log \text{psDSize}^M(\mathcal{S}), n) \cdot \min(\log m, \log \text{psDSize}^M(\mathcal{S})))$ .

**Proof Sketch.** We give a proof sketch for the second statement. For search problem  $\mathcal{S}$  with  $\text{psDSize}^M(\mathcal{S}) = s$ , let function  $\tilde{f}$  solve  $\mathcal{S}$  with  $\text{RSize}^M(\tilde{f}) = s$  witnessed by a randomized tree  $T$ . For  $k = \lceil \log m \rceil$ , define the Boolean functions  $f_1, f_2, \dots, f_k$  where  $f_i(x)$  extracts the  $i$ th bit in the  $k$ -bit representation of  $\tilde{f}(x)$ . Then for each  $i \in [k]$ ,  $\text{RSize}^M(f_i) \leq \text{RSize}^M(\tilde{f}) = s$ ; simply relabel the leaves of  $T$  appropriately. By the hypothesised relation for Boolean functions, for each  $i \in [k]$ ,  $\log \text{DSize}^M(f_i) \leq q(\log \text{RSize}^M(f_i), n) \leq q(\log s, n)$ . Let  $T_i$  be a deterministic tree achieving this size bound. By composing the trees  $T_1, T_2, \dots, T_k$  and suitably relabelling the leaves with elements from  $[m]$ , we obtain a deterministic tree for  $\tilde{f}$  of size  $(2^{q(\log s, n)})^k$ . With careful analysis, we can get  $\min(\log m, \log \text{psDSize}^M(\mathcal{S}))$  in place of  $k$  in the exponent. ◀

Recently it was shown in [4] that for total Boolean functions, the logarithms of deterministic and randomized size are polynomially related, ignoring a polylog factor in input size.

► **Proposition 4.2** ([4, Theorem 3.1(b)]). For a total Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,

$$\log \text{DSize}^{\text{dt}}(f) = O((\log \text{RSize}^{\text{dt}}(f))^4 \log^3(n)).$$

Using the relation from Proposition 4.2 as the “hypothesised function” in Proposition 4.1(2), we obtain the following result, relating the log of deterministic size and the log of pseudo-deterministic size for search problems.

► **Theorem 4.3.** *For a total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ , we have*

$$\log \text{RSize}^{\text{dt}}(\mathcal{S}) = O(\log^4 \text{psDSize}^{\text{dt}}(\mathcal{S}) \cdot \log^3(n) \cdot \min(\log m, \log \text{psDSize}^{\text{dt}}(\mathcal{S}))).$$

In [9] (Theorem 22), a separation was established between pseudo-deterministic and randomized size for a SearchCNF problem, defined on suitably expanding kCNF formulas *lifted* with 2-bit XOR gadgets. It was shown that the randomized size complexity of this problem is  $O(1)$ , while the pseudo-deterministic size complexity is  $\exp(\Omega(\sqrt{n}))$ . We obtain a similar (but weaker) separation for the SearchCNF problem *without any lifting*, by putting together Proposition 2.5(Item 3) and Theorem 4.3.

► **Corollary 4.4.** *For  $F$  a random 3-CNF formula on  $n$  variables with  $m = cn$  clauses sampled from  $\mathcal{F}_m^{3,n}$ , with probability  $1 - o(1)$ ,  $F$  is unsatisfiable and*

$$\text{psDSize}^{\text{dt}}(\text{SEARCHCNF}(F)) = \exp(\Omega(n^{1/4}/\log n)).$$

Since  $\text{RSize}^{\text{dt}}$  of SEARCHCNF on random 3-CNF formulas is  $O(1)$  w.h.p (see Proposition 2.5(Item 1)), we get a separation between  $\text{RSize}^{\text{dt}}$  and  $\text{psDSize}^{\text{dt}}$ , albeit not as strong as [9]. However, Theorem 4.3 allows us to conclude that any total search problem separating randomized and deterministic size will yield a separation between  $\text{RSize}^{\text{dt}}$  and  $\text{psDSize}^{\text{dt}}$ .

We now improve the separation between randomized and pseudo-deterministic sizes, from  $O(1)$  vs  $\exp(\Omega(\sqrt{n}))$  as shown in [9], to  $O(1)$  vs  $\exp(\Omega(n))$ . To achieve this, we focus on the APPROXHAWWT problem. For this problem, a linear depth separation between randomized and pseudo-deterministic algorithms is already known from [8] (see Proposition 2.4). By using a 1-bit indexing gadget, we can lift the depth separation in APPROXHAWWT to an exponential size separation, as was done in [9, Theorem 22]. (The 1-bit indexing gadget replaces each variable  $x$  by the function  $\text{SEL}(x^a, x^b, x^c) = \text{if } x^a = 1 \text{ then } x^b \text{ else } x^c$ .) In the rest of this section, we show that the exponential size separation between randomized and pseudo-deterministic algorithms can also be achieved using APPROXHAWWT itself *without the lift*. It is easy to see that the randomized size of APPROXHAWWT is  $O(1)$ . We show that its pseudo-deterministic size is  $\exp(\Omega(n))$ . To this end, we establish that every solution to APPROXHAWWT embeds a hard boolean function whose randomized decision tree size is exponential in the input size. This hard function is a completion of the promise problem Approximate Majority, APPROXMAJ.

APPROXMAJ is a promise problem (i.e. partial Boolean functions; certain bit strings are promised to never appear as inputs) where the task is to compute the majority of the given bit string. The promise is that the fraction of bits set to 1 in the input is either at least  $3/4$  or at most  $1/2$ . A completion of APPROXMAJ is a total Boolean function that extends APPROXMAJ arbitrarily on the non-promised inputs. We show that every solution to APPROXHAWWT embeds some completion of APPROXMAJ, and that the randomized decision tree size of every completion of APPROXMAJ is exponential in the input size.

► **Theorem 4.5.** *For the promise problem (partial boolean function) APPROXMAJ,*

$$\text{APPROXMAJ}(x) = \begin{cases} 0 & \text{if } |x| \leq n/2 \\ 1 & \text{if } |x| \geq 3n/4 \end{cases},$$

*every completion  $f$  of APPROXMAJ has  $\text{RSize}^{\text{dt}}(f) = \exp(\Omega(n))$ .*

The proof of Theorem 4.5 is based on a corruption argument and follows the template for proving randomized decision tree size lower bounds in [4, Theorem A.7]. This argument is essentially due to Swagato Sanyal, and we thank him for allowing us to include it here. Before we see its proof, let us use it to establish an exponential separation between randomized and pseudo-deterministic size for APPROXHAWWT.

► **Theorem 4.6.** *Let  $\mathcal{S}$  be the search problem  $\text{APPROXHAWWT} = \{(x, v) \in \{0, 1\}^n \times \{0, 1, \dots, n\} : |wt(x) - v| \leq n/10\}$ , where  $wt(x)$  is the Hamming weight of  $x$ . Then  $\text{RSize}^{\text{dt}}(\mathcal{S}) = O(1)$ , while  $\text{psDSize}^{\text{dt}}(\mathcal{S}) = \exp(\Omega(n))$ .*

**Proof.** The Hamming weight of a string can be estimated within a small ( $\theta(n)$ ) additive error by querying a constant number of variables uniformly at random and outputting the scaled-up fraction of 1's seen in the queried bits. Thus  $\text{RSize}^{\text{dt}}(\mathcal{S}) = O(1)$ .

To show a pseudo-deterministic size lower bound, we need to show that any function that solves the  $\text{APPROXHAWWT}$  problem must have randomized decision tree of size  $\exp(\Omega(n))$ . Let  $f$  be any function solving the  $\text{APPROXHAWWT}$  problem. The total Boolean function  $\bar{f} : \{0, 1\}^n \rightarrow \{0, 1\}$  defined as  $\bar{f}(x) = 1$  iff  $f(x) > 6n/10$  is a completion of  $\text{APPROXMAJ}$ . Given a randomized decision tree that computes  $f$ , we can relabel the leaves appropriately to obtain a randomized decision tree that computes  $\bar{f}$ . Using Theorem 4.5, we conclude that  $\text{RSize}^{\text{dt}}(f) \geq \text{RSize}^{\text{dt}}(\bar{f}) = \exp(\Omega(n))$ . ◀

**Proof of Theorem 4.5.** Let  $f$  be any completion of  $\text{APPROXMAJ}$ . Our strategy is to construct a hard distribution  $\mathcal{D}$  on the inputs  $\{0, 1\}^n$  such that  $\text{DSize}_{\mathcal{D}, 1/3}^{\text{dt}}(f) = \exp(\Omega(n))$ , and then use Yao's minmax principle (see Proposition 2.1) to conclude that  $\text{RSize}^{\text{dt}}(f) = \exp(\Omega(n))$ . To define the hard distribution, we start by introducing some terminology. For an input  $x \in \{0, 1\}^n$ , let  $S_x^1 = \{i : x_i = 1\}$  and  $S_x^0 = [n] \setminus S_x^1$ . We say that  $x$  is 0-sensitive if all the 0s in  $x$  are sensitive with respect to  $f$ . For  $x \in \{0, 1\}^n$ , we define the set of extreme upward neighbors of  $x$  as  $\text{EUN}(x) = \{y : S_x^1 \subseteq S_y^1, f(x) = f(y) \text{ and } y \text{ is 0-sensitive}\}$ . With this terminology in place, we can define the hard distribution as follows:

1. Let  $\text{rep} : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a function which maps  $x \in \{0, 1\}^n$  to an arbitrary input from  $\text{EUN}(x)$ . Define  $\mu_0$ , a distribution over  $f^{-1}(0)$  as follows: Sample an  $x$  of Hamming weight  $n/2$  uniformly at random, and output  $\text{rep}(x)$ .
2. Define  $\mu_1$ , a distribution over  $f^{-1}(1)$ , as follows: Sample an  $x$  according to  $\mu_0$ , an index  $i$  uniformly at random from  $S_x^0$ , and return  $x \oplus 1_{\{i\}}$ .
3. Our hard distribution  $\mathcal{D}$  is  $(\mu_0 + \mu_1)/2$  i.e. with probability  $1/2$  return a sample from  $\mu_0$ , and with probability  $1/2$  return a sample from  $\mu_1$ .

We show below that  $\text{DSize}_{\mathcal{D}, 1/10}^{\text{dt}}(f) = \exp(\Omega(n))$ . Let  $T$  be a deterministic decision tree computing  $f$  correctly on at least  $9/10$ -probability mass when the input is sampled according to  $\mathcal{D}$ . Since  $\mathcal{D}$  samples with probability  $1/2$  from  $\mu_0$  and with probability  $1/2$  from  $\mu_1$ ,  $T$  must be correct on at least  $4/5$ -th mass of  $\mu_0$  as well as at least  $4/5$  mass of  $\mu_1$ . Let  $L_0$  be set of all 0-labelled leaves (0-leaves) in  $T$ . Let  $\rho_0$  and  $\rho_1$  be the  $\mu_0$  and  $\mu_1$  mass captured by 0-leaves respectively; i.e.,

$$\rho_0 = \sum_{v \in L_0} \Pr_{x \sim \mu_0} [x \text{ reaches } v]; \quad \rho_1 = \sum_{v \in L_0} \Pr_{x \sim \mu_1} [x \text{ reaches } v].$$

As discussed above,  $\rho_0 \geq 4/5$  and  $\rho_1 \leq 1/5$ .

For a leaf  $v$ , let  $Z_v$  denote the set of indices of variables fixed to zero on the path leading to  $v$  and  $E_v(x)$  denote the event that the input  $x$  reaches leaf  $v$ . We will show that 0-paths with small  $|Z_v|$  together capture at most  $2/5$  of the  $\mu_0$  mass, and 0-paths with large  $|Z_v|$  individually capture exponentially small  $\mu_0$  mass. Thus to ensure that  $\rho_0$  is large enough, there must be many 0-leaves.

1. (0-paths with few 0's). Firstly, we show that 0-paths which see less than  $\lceil n/8 \rceil$  0's must capture no more than  $2/5$ -th mass of  $\mu_0$ , i.e.,

$$\rho_0^0 = \sum_{\substack{v \in L_0 \\ |Z_v| < \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [x \text{ reaches } v] \leq 2/5.$$

## 34:10 Query Complexity of Search Problems

This follows from the sensitivity property of  $\mu_0$ . Specifically, each  $y$  in the support of  $\mu_0$  has a Hamming weight less than  $3n/4$ , and since all the 0s in  $y$  are sensitive, each  $y$  in the support of  $\mu_0$  has 0-sensitivity of at least  $n/4$ . Therefore, if a 0-path has not observed many 0s, the corresponding leaf will also capture a significant amount of  $\mu_1$  mass. Formally, consider a subcube  $Q$  corresponding to a 0-leaf with less than  $\lceil n/8 \rceil$  variables fixed to 0. Due to the sensitivity property of  $\mu_0$ , each  $x$  supported by  $\mu_0$  has at least  $n/4$  sensitive 0s. Hence, any  $x$  supported by  $\mu_0$  that lies in  $Q$  has at least half of its total 0s unfixed. By flipping any of these 0s, we obtain an input supported by  $\mu_1$  that still lies in  $Q$ . Therefore,

$$\begin{aligned} 1/5 \geq \rho_1 &\geq \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{x \sim \mu_1} [E_v(x)] = \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{\substack{x \sim \mu_0 \\ i \sim_u S_x^0}} [E_v(x \oplus 1_{\{i\}})] \\ &\geq \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{\substack{x \sim \mu_0 \\ i \sim_u S_x^0}} [E_v(x) \text{ and } i \notin Z_v] = \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [E_v(x)] \Pr_{i \sim_u S_x^0} [i \notin Z_v | E_v(x)] \\ &\geq \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [E_v(x)] \cdot \left( \frac{|S_x^0| - n/8}{|S_x^0|} \right) \geq \frac{1}{2} \cdot \left( \sum_{\substack{v \in L_0 \\ |v|_0 < \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [E_v(x)] \right) = \frac{1}{2} \rho_0^0. \end{aligned}$$

(In the last inequality, we use the fact that  $|S_x^0| > n/4$ .) Hence  $\rho_0^0 \leq 2/5$ .

2. (0-paths with lots of 0's). Secondly, we show that a 0-path which sees more than  $\lceil n/8 \rceil$  0's can capture at most  $\kappa = \exp(-\Omega(n))$  of  $\mu_0$  mass. Consider a leaf  $v$  labelled 0 such that the path leading to  $v$  fixes  $t \geq \lceil n/8 \rceil$  variables to 0;  $|Z_v| = t \geq n/8$ . Let  $S_{n/2}$  be all strings of Hamming weight  $n/2$ . We have

$$\begin{aligned} \kappa &= \Pr_{y \sim \mu_0} [y \text{ reaches } v] = \Pr_{x \sim_u S_{n/2}} [rep(x) \text{ reaches } v] \\ &\leq \Pr_{x \sim_u S_{n/2}} [S_{rep(x)}^1 \cap Z_v = \emptyset] \leq \Pr_{x \sim_u S_{n/2}} [S_x^1 \cap Z_v = \emptyset] \quad (\text{because } S_x^1 \subseteq S_{rep(x)}^1) \\ &\leq \frac{\binom{n-t}{n/2}}{\binom{n}{n/2}} \leq \frac{\binom{7n/8}{n/2}}{\binom{n}{n/2}} = \prod_{i=0}^{n/2-1} \frac{7n/8 - i}{n - i} \leq \left( \frac{7}{8} \right)^{n/2} = 2^{-\Omega(n)}. \end{aligned}$$

With these two observations, we can now obtain the desired lower bound.

$$\begin{aligned} 4/5 \leq \rho_0 &= \sum_{v \in L_0} \Pr_{x \sim \mu_0} [E_v(x)] = \sum_{\substack{v \in L_0 \\ |Z_v| < \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [E_v(x)] + \sum_{\substack{v \in L_0 \\ |Z_v| \geq \lceil n/8 \rceil}} \Pr_{x \sim \mu_0} [E_v(x)] \\ &\leq 2/5 + \kappa \times (\text{number of 0-leaves}). \end{aligned}$$

Hence the number of 0-leaves is at least  $2/(5\kappa) = \exp(\Omega(n))$ . ◀

## 5 More general decision trees

A variable is queried at each node of a decision tree. Generalising the class of permitted queries gives rise to many variants of decision trees that have been considered in different contexts. In this section, we consider three such classes.

- *AND-decision trees (ADT)*: queries are restricted to AND of non-negated variables.
- *(AND, OR)-decision trees*: queries are restricted to AND or OR of variables.
- *PARITY-decision trees (PDT)*: queries are restricted to PARITY of variables.

We denote the deterministic, pseudo-deterministic and randomized query complexity in each of these types of trees as  $(D^{\wedge\text{-dt}}, \text{psD}^{\wedge\text{-dt}}, R^{\wedge\text{-dt}})$ ,  $(D^{(\wedge, \vee)\text{-dt}}, \text{psD}^{(\wedge, \vee)\text{-dt}}, R^{(\wedge, \vee)\text{-dt}})$ , and  $(D^{\oplus\text{-dt}}, \text{psD}^{\oplus\text{-dt}}, R^{\oplus\text{-dt}})$ , respectively.

AND, OR and PARITY are the most basic Boolean functions. It is thus natural to study the relationship between determinism, pseudo-determinism and randomization in these settings. We will use the following recently-proved results from [4].

► **Proposition 5.1** ([4]). *For every total Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ ,*

1.  $D^{\wedge\text{-dt}}(f) = O(R^{\wedge\text{-dt}}(f)^3 \log^4(n))$ . ([4, Theorem 4.5])
2.  $\log \text{DSize}^{\text{dt}}(f)/(2 \log n) \leq D^{(\wedge, \vee)\text{-dt}}(f) \leq 4 \log \text{DSize}^{\text{dt}}(f)$ . ([4, Lemma 4.2])
3.  $\log \text{RSize}^{\text{dt}}(f)/(2 \log n) \leq R^{(\wedge, \vee)\text{-dt}}(f) \leq 4 \log \text{RSize}^{\text{dt}}(f)$ . ([4, Lemma 4.2])
4.  $D^{(\wedge, \vee)\text{-dt}}(f) = O(R^{(\wedge, \vee)\text{-dt}}(f)^4 \log^7(n))$ . ([4, Theorem 4.1])

### AND-decision trees

Pseudo-determinism can be separated from randomness in AND decision trees. To establish the separation, we first give a technique to prove a pseudo-deterministic lower bound using 0-block sensitivity. The following theorem generalizes Theorem 3.1(2) to AND decision trees. The same relation is proved for Boolean functions in [12], by reduction to a hard communication problem; We give a more direct proof in the full version of our paper [3] for multi-output functions by constructing a hard distribution and using Yao's minimax principle.

► **Theorem 5.2.** *For a multi-output function  $f$ ,  $R_{1/3}^{\wedge\text{-dt}}(f) \geq bs_0(f)/3$ .*

*For a total search problem  $\mathcal{S}$ ,  $\text{psD}_{1/3}^{\wedge\text{-dt}}(\mathcal{S}) \geq \min_{f \in \mathcal{S}} bs_0(f)/3$ .*

Using this result, we can now separate randomized and pseudo-deterministic complexity.

► **Theorem 5.3.** *Let  $\mathcal{S}$  be the search problem  $\text{APPROXHAMWT} = \{(x, v) : |wt(x) - v| \leq n/10\}$ . Then  $R^{\wedge\text{-dt}}(\mathcal{S}) = R^{\text{dt}}(\mathcal{S}) = O(1)$ , while  $\text{psD}^{\wedge\text{-dt}}(\mathcal{S}) = \Omega(n)$ .*

**Proof.** It is easy to see, and already noted in Corollary 4.2 of [8], that  $R^{\text{dt}}(\mathcal{S}) = O(1)$ . To show  $\text{psD}^{\wedge\text{-dt}}(\mathcal{S}) = \Omega(n)$ , we will show that any  $f$  solving  $\mathcal{S}$  must have 0-sensitivity of at least  $4n/5$ . This too follows the proof outline from Corollary 4.2 of [8], where a lower bound on  $\text{psD}^{\text{dt}}$  was obtained. But using Theorem 5.2, we draw the stronger conclusion that  $\text{psD}^{\wedge\text{-dt}}(\mathcal{S}) \geq 4n/5$ . Suppose that for some  $f$  solving  $\mathcal{S}$ ,  $s_0(f) < 4n/5$ . We start with  $x^0 = 0^n$  and create a sequence of inputs  $\langle x^i \rangle$  such that  $wt(x^i) = i$  and  $f(x^i) = f(0^n)$ . Because  $f$  solves APPROXHAMWT,  $n/10 \geq f(0^n) = f(x^1) = f(x^2) = \dots = f(x^l) \geq l - n/10$ . Thus if we are able to create such a sequence of length at least  $l = n/5 + 1$ , then we already have a contradiction. The only thing left is to create the sequence  $x^i$ . For  $0 \leq i \leq n/5$ , given  $x^i$  with  $f(x^i) = f(0^n)$ , we need to find a suitable  $x^{i+1}$ . Note that  $x^i$  has exactly  $n - i$  0-bit positions, of which at most  $s_0(f)$  are sensitive, so at least  $s = n - i - s_0(f)$  0-bit positions are not sensitive. Since  $s_0(f) < 4n/5$  and  $i \leq n/5$ ,  $s > 0$ , so  $x^i$  has at least one non-sensitive 0-bit position. Pick any such position, say  $j$ , and define  $x^{i+1} = x^i \oplus 1_{\{j\}}$ . Note that  $x^{i+1}$  satisfies the desired properties i.e.  $f(x^{i+1}) = f(x^i) = f(0^n)$  and  $wt(x^{i+1}) = i + 1$ . ◀

On the other hand, using Proposition 5.1(1) with Proposition 4.1, we get a polynomial relationship between  $\text{psD}^{\wedge\text{-dt}}$  and  $D^{\wedge\text{-dt}}$ .

► **Theorem 5.4.** *For a total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ , we have*

$$D^{\wedge\text{-dt}}(\mathcal{S}) = O(\text{psD}^{\wedge\text{-dt}}(\mathcal{S})^3 \cdot \log^4(n) \cdot \min(\log m, \text{psD}^{\wedge\text{-dt}}(\mathcal{S}))).$$

**(AND, OR)-decision trees**

The results of Proposition 5.1(2),(3) are proved in [4] only for total Boolean functions. However, the proof there is based on a syntactic argument, where the upper bound relies on a tree-balancing argument and the lower bound is obtained by opening up AND and OR queries. Since the proof is syntactic, it naturally extends to multi-output functions and search problems as well. Using this extension to multi-output functions, we obtain the following relationship between  $\text{psDSize}^{\text{dt}}$  and  $\text{psD}^{(\wedge, \vee)\text{-dt}}$ .

► **Lemma 5.5.** *For a total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ , we have*

$$\log \text{psDSize}^{\text{dt}}(\mathcal{S}) / (2 \log n) \leq \text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) \leq 4 \log \text{psDSize}^{\text{dt}}(\mathcal{S}).$$

**Proof.** For  $\mathcal{S}$ , let  $f$  and  $g$  be multi-output function solving  $\mathcal{S}$ , with  $\text{psDSize}^{\text{dt}}(\mathcal{S}) = \text{RSize}^{\text{dt}}(f)$  and  $\text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) = \text{R}^{(\wedge, \vee)\text{-dt}}(g)$  respectively. Then

$$\begin{aligned} 4 \log \text{psDSize}^{\text{dt}}(\mathcal{S}) &= 4 \log \text{RSize}^{\text{dt}}(f) \stackrel{(*)}{\geq} \text{R}^{(\wedge, \vee)\text{-dt}}(f) \geq \text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) \\ &= \text{R}^{(\wedge, \vee)\text{-dt}}(g) \stackrel{(*)}{\geq} \log \text{RSize}^{\text{dt}}(g) / (2 \log n) \geq \log \text{psDSize}^{\text{dt}}(\mathcal{S}) / (2 \log n). \end{aligned}$$

The inequalities marked (\*) holds because of Proposition 5.1(3). ◀

This, along with the size separation from Theorem 4.6, gives us a separation between randomized and pseudo-deterministic query complexity in (AND, OR)-decision trees.

► **Theorem 5.6.** *Let  $\mathcal{S}$  be the search problem APPROXHAMWT. Then  $\text{R}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) = O(1)$  and  $\text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) = \Omega(n / \log n)$ .*

On the other hand, using Proposition 5.1(4) along with Proposition 4.1 gives a polynomial relation between pseudo-determinism and determinism, upto polylog  $n$  factors.

► **Theorem 5.7.** *For a total search problem  $\mathcal{S} \subseteq \{0, 1\}^n \times [m]$ , we have*

$$\text{D}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}) = O(\text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S})^4 \cdot \log^7(n) \cdot \min(\log m, \text{psD}^{(\wedge, \vee)\text{-dt}}(\mathcal{S}))).$$

**PARITY-decision trees**

For PARITY decision trees, for total Boolean functions, the randomized and deterministic PARITY query complexities are linearly separated: for the AND and OR functions, the deterministic PDT complexity is  $\Omega(n)$ , whereas the randomized PDT complexity is  $O(1)$ . The search analogue of the OR function gives an almost linear separation between determinism and pseudo-determinism in the PDT model. See the full version of our paper [3] for details.

► **Theorem 5.8.** *Let  $\mathcal{S}$  be the search problem SEARCHOR =  $\{(x, v) : (x_v = 1) \text{ or } (x = 0^n \wedge v = n + 1)\}$ . Then  $\text{D}^{\oplus\text{-dt}}(\mathcal{S}) = n$  whereas  $\text{psD}^{\oplus\text{-dt}}(\mathcal{S}) = O(\log n \log \log n)$ .*

Establishing a super-polynomial separation between randomness and pseudo-determinism remains open for PARITY decision trees.

**6 A combinatorial proof of a combinatorial problem**

In [9], the authors studied the pseudo-deterministic query complexity of a promise problem (PROMISEFIND1). Here the input bit string has 1s in at least half the positions, and the task is to find a 1. They observed that PROMISEFIND1 is a complete problem for easily-verifiable

search problems with randomized query algorithms (see Theorem 3 in [9]), and proved a  $\Omega(\sqrt{n})$  lower bound on its pseudo-deterministic query complexity. They conjectured that the pseudo-deterministic query lower bound for PROMISEFIND1 can be improved to  $\Omega(n)$ . Towards understanding the PROMISEFIND1 problem better, they introduced a natural colouring problem on hypercubes which states that any proper coloring of the hypercube contains a point with many 1s and with high block sensitivity.

► **Definition 6.1.** *A proper coloring of the  $n$ -dimensional hypercube  $H_n$  is any function  $\phi : \{0, 1\}^n - \{0^n\} \rightarrow [n]$  such that for all  $\beta \in \{0, 1\}^n - \{0^n\}$ ,  $\beta_{\phi(\beta)} = 1$ .*

We say a proper coloring  $\phi$  is  $d$ -sensitive if there exists a  $\beta \in \{0, 1\}^n$  such that  $|\beta|_1 \geq n/2$  and  $\beta$  has block sensitivity at least  $d$  with respect to  $\phi$ . That is, there are  $d$  disjoint blocks of inputs,  $B_1, \dots, B_d$  such that for all  $i \in [d]$ ,  $\phi(\beta) \neq \phi(\beta \oplus 1_{B_i})$ . The hypercube coloring problem is about proving lower bound on the (block) sensitivity of every proper coloring. In [9] it was shown that every proper coloring is  $\Omega(\sqrt{n})$ -sensitive.

► **Proposition 6.2** (Theorem 14 [9]). *Every proper coloring of  $H_n$  is  $\Omega(\sqrt{n})$ -sensitive.*

The hypercube coloring problem is closely related to the pseudodeterministic query complexity of PROMISEFIND1. It is a straightforward observation that showing every proper coloring is  $d$ -sensitive implies a lower bound of  $d$  on the pseudo-deterministic query complexity of PROMISEFIND1. To prove Proposition 6.2 in [9], the sensitivity lower bound for the search problem associated with a random unsat  $k$ -XOR formula was converted into a block sensitivity lower bound for the hypercube coloring problem.

We give a self-contained combinatorial solution to the coloring problem. Our solution shows that every proper coloring of hypercube has a  $\beta \in \{0, 1\}^n$  with Hamming weight  $\geq n/2$  and with block sensitivity  $\Omega(n^{1/3})$ . In fact, either the 1-block sensitivity or the 0-block sensitivity (or both) is  $\Omega(n^{1/3})$ . Thus this appears incomparable with the bound from [9]. Our solution is constructive: Algorithm 1 finds the required high-weight, high-block-sensitivity point, by querying  $\phi$  at various points.

► **Theorem 6.3.** *Every proper coloring  $\phi$  of the Boolean hypercube has a  $\beta \in \{0, 1\}^n$  with  $|\beta| \geq n/2$  satisfying  $bs_0(\phi, \beta) = \Omega(n^{1/3})$  or  $bs_1(\phi, \beta) = \Omega(n^{1/3})$ .*

**Proof.** In Algorithm 1, we describe a procedure to find the required point  $\beta$ . To prove that the algorithm is correct, we need to prove that if it returns  $\beta \in \{0, 1\}^n$  and blocks  $D_1, D_2, \dots, D_r$ , then

1.  $\beta \in \mathcal{X}$  (i.e.  $\beta$  has Hamming weight at least  $n/2$ ),
2.  $D_1, D_2, \dots, D_r$  are disjoint sensitive blocks of  $\phi$  at  $\beta$ , and
3. either all these blocks are 1-blocks of  $\beta$  or all these blocks are 0-blocks.
4.  $r = \Omega(n^{1/3})$ ,

Observe that by construction, for each  $i \in [t+1]$  where  $\beta^i$  is constructed by the algorithm,  $\beta^i$  has 0s in  $B_j$  for  $j < i$  and 1s in  $B_i$  (in fact, 1s elsewhere); hence the blocks  $B_1, \dots, B_{i-1}$  are disjoint.

Further, by construction, each complete iteration of the for loop adds fewer than  $t^2$  positions to  $C$ : there are fewer than  $t$  blocks (otherwise the algorithm would terminate at line 9) and each block has size less than  $t$  (otherwise the algorithm would terminate at line 13). Thus, since  $|C_0| = 0$ , if the algorithm reaches line 15 in iteration  $i$ , then  $C_i$  has size less than  $i \cdot t^2$ . Hence  $\beta^{i+1}$  has hamming weight  $n - |C_i| > n - it^2 \geq n - t^3 > n - n/2 \geq n/2$  and is in  $\mathcal{X}$ .

■ **Algorithm 1** Algorithm to find the sensitive point.

---

**Require:** A proper coloring  $\phi$ . i.e. For  $\mathcal{X} = \{x \in \{0, 1\}^n \mid \sum_i x_i \geq n/2\}$ ,  $\phi : \mathcal{X} \rightarrow [n]$  satisfying  $\forall x \in \mathcal{X}, x_{\phi(x)} = 1$ .

- 1:  $t \leftarrow \lfloor (n/2)^{1/3} \rfloor$ ;  $C_0 \leftarrow \emptyset$
- 2: **for**  $i$  from 1 to  $t$  **do**
- 3:      $\beta^i \leftarrow 0_{C_{i-1}}$  ▷ Reference input to find  $t$  sensitive 1-blocks.
- 4:      $\ell \leftarrow \phi(\beta^i)$ ;  $s \leftarrow \text{bs}_1(\phi, \beta^i)$  ▷  $\{\ell\}$  is a 1-sensitive block of  $\beta^i$ , so  $s \geq 1$
- 5:      $B_{i,1}, B_{i,2}, \dots, B_{i,s}$ : disjoint, minimally-sensitive
- 6:     1-blocks achieving the 1-block sensitivity  $s$ .
- 7:      $B_i \leftarrow \cup_{j=1}^s B_{i,j}$  ▷  $\ell \in B_i$
- 8:     **if**  $s \geq t$  **then**
- 9:         **return**  $\beta^i$  and  $\{B_{i,1}, B_{i,2}, \dots, B_{i,s}\}$  ▷  $\text{bs}_1(\phi, \beta^i) \geq t$
- 10:     **end if**
- 11:     **if**  $\max_{j \in [s]} |B_{i,j}| \geq t$  **then**
- 12:         Pick any such  $j \in [s]$  with  $|B_{i,j}| \geq t$ .
- 13:         **return**  $\beta^i \oplus 1_{B_{i,j}}$  and  $\{\{k\} \mid k \in B_{i,j}\}$  ▷  $s_0(\phi, \beta^i \oplus 1_{B_{i,j}}) \geq t$
- 14:     **end if**
- 15:      $C_i \leftarrow C_{i-1} \cup B_i$  ▷ We show:  $C_i$  forms a  $\phi$ -certificate for  $\beta^i$
- 16: **end for**
- 17:  $\beta^{t+1} \leftarrow 0_{C_t}$
- 18: **return**  $\beta^{t+1}$  and  $\{B_1, B_2, \dots, B_t\}$  ▷  $\text{bs}_0(\phi, \beta^{t+1}) \geq t$

---

If the algorithm terminates at line 9 in the  $i$ th iteration of the for loop, then by the choice in line 6 the returned blocks are disjoint 1-sensitive blocks of  $\beta = \beta^i$ , and there are at least  $t$  of them. Similarly, if the algorithm terminates at line 13 in the  $i$ th iteration of the for loop, then by minimality of the sensitive block  $B_{i,j}$  chosen in line 12, each position in  $B_{i,j}$  is a 0-sensitive location in  $\beta = \beta^i \oplus 1_{B_{i,j}}$ , and there are at least  $t$  of them.

If the algorithm terminates at line 18, then each  $B_i$  is a 0-block of  $\beta = \beta^{t+1}$  and there are  $t$  such blocks. It remains to prove that each  $B_i$  is sensitive for  $\beta = \beta^{t+1}$ . To show this, we will first show that each  $C_i$  is a certificate for  $\beta^i$ , and then show that this implies each  $B_i$  is sensitive for  $\beta$ .

For the first part, suppose for some  $i \in [t]$ ,  $C_i$  is not a certificate for  $\beta^i$ . Then there exists an  $\alpha \in \mathcal{X}$  such that  $\forall j \in C_i, \alpha_j = \beta_j^i$ , but  $\phi(\alpha) \neq \phi(\beta^i)$ . Let  $B$  be the set of positions where  $\alpha$  and  $\beta^i$  differ i.e.  $\alpha = \beta^i \oplus 1_B$ . Since  $\alpha$  and  $\beta^i$  agree on  $C_i$ ,  $B$  must be disjoint from  $C_i$ . Since  $\phi(\beta^i) \neq \phi(\alpha) = \phi(\beta^i \oplus 1_B)$ ,  $B$  is a 1-sensitive block of  $\phi$  at  $\beta^i$ . By the choice in line 6 at the  $i$ th iteration,  $\beta^i$  has no 1-sensitive blocks disjoint from the blocks  $B_{i,1}, \dots, B_{i,s}$ . But  $B_i$  is precisely the union of the these blocks, and is contained in  $C_i$ , so  $B$  is disjoint from  $B_i$ , a contradiction. Hence  $C_i$  is indeed a  $\phi$ -certificate for  $\beta^i$ .

For the second part, note that for each  $i \in [t]$ ,  $\beta$  and  $\beta^i$  agree on  $C_{i-1}$  and  $\beta \oplus B_i$  and  $\beta^i$  agree on  $C_i$ . Since  $C_i$  is a certificate for  $\beta^i$ ,  $\phi(\beta \oplus B_i) = \phi(\beta^i) = \ell$ , say. By the definition of proper coloring,  $\{\ell\}$  is a 1-sensitive block of  $\beta^i$ , and since the blocks chosen in line 6 are the maximum possible 1-sensitive blocks,  $\ell \in B_i$ . But  $\phi(\beta) \neq \ell$  because  $\beta = 0_{C_t}$  and has only 0s in  $B_i$ . Thus  $\phi(\beta) \neq \phi(\beta \oplus B_i)$ , and hence  $B_i$  is a 0-sensitive block for  $\beta$ .

Finally, by choice of  $t$ , we see that  $r = \Omega(n^{1/3})$ . This concludes the correctness proof. ◀

---

**References**

---

- 1 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow – Resolution made simple. *J. ACM*, 48(2):149–169, 2001. doi:10.1145/375827.375835.
- 2 Harry Buhman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
- 3 Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan. Query complexity of search problems. *Electron. Colloquium Comput. Complex.*, TR23-039, 2023. URL: <https://eccc.weizmann.ac.il/report/2023/039/>.
- 4 Arkadev Chattopadhyay, Yogesh Dahiya, Nikhil S Mande, Jaikumar Radhakrishnan, and Swagato Sanyal. Randomized versus deterministic decision tree size. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 867–880, 2023.
- 5 Daniel Dadush and Samarth Tiwari. On the Complexity of Branching Proofs. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:35, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2020.34.
- 6 Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–911, 2018.
- 7 Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, page 136, 2011.
- 8 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science ITCS*, pages 127–138. ACM, 2013. See also ECCC Vol. 19, T.R. 12-101, 2012.
- 9 Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the pseudo-deterministic query complexity of NP search problems. In Valentine Kabanets, editor, *36th Computational Complexity Conference CCC*, volume 200 of *LIPIcs*, pages 36:1–36:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 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 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Guest column: Models of computation between decision trees and communication. *ACM SIGACT News*, 52(2):46–70, 2021.
- 12 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 197–208, 2021.
- 13 László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM Journal on Discrete Mathematics*, 8(1):119–132, 1995.
- 14 Noam Nisan. Crew prams and decision trees. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 327–335, 1989.
- 15 Andrew Chi-Chih Yao. Lower bounds by probabilistic arguments (extended abstract). In *24th Annual Symposium on Foundations of Computer Science FOCS*, pages 420–428. IEEE Computer Society, 1983.



# Tight Algorithmic Applications of Clique-Width Generalizations

Vera Chekan  

Humboldt-Universität zu Berlin, Germany

Stefan Kratsch  

Humboldt-Universität zu Berlin, Germany

---

## Abstract

In this work, we study two natural generalizations of clique-width introduced by Martin Fürer. Multi-clique-width (mcw) allows every vertex to hold multiple labels [ITCS 2017], while for fusion-width (fw) we have a possibility to merge all vertices of a certain label [LATIN 2014]. Fürer has shown that both parameters are upper-bounded by treewidth thus making them more appealing from an algorithmic perspective than clique-width and asked for applications of these parameters for problem solving. First, we determine the relation between these two parameters by showing that  $mcw \leq fw + 1$ . Then we show that when parameterized by multi-clique-width, many problems (e.g., CONNECTED DOMINATING SET) admit algorithms with the same running time as for clique-width despite the exponential gap between these two parameters. For some problems (e.g., HAMILTONIAN CYCLE) we show an analogous result for fusion-width: For this we present an alternative view on fusion-width by introducing so-called glue-expressions which might be interesting on their own. All algorithms obtained in this work are tight up to (Strong) Exponential Time Hypothesis.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms

**Keywords and phrases** Parameterized complexity, connectivity problems, clique-width

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.35

**Related Version** *Full Version:* <https://arxiv.org/abs/2307.04628> [8]

**Funding** *Vera Chekan:* Supported by DFG Research Training Group 2434 “Facets of Complexity”.

## 1 Introduction

In parameterized complexity apart from the input size we consider a so-called parameter and study the complexity of problems depending on both the input size and the parameter where the allowed dependency on the input size is polynomial. In a more fine-grained setting one is interested in the best possible dependency on the parameter under reasonable conjectures. A broad line of research is devoted to so-called *structural parameters* measuring how simple the graph structure is: different parameters quantify various notions of possibly useful input structure. Probably the most prominent structural parameter is treewidth, which reflects how well a graph can be decomposed using small vertex separators. For a variety of problems, the tight complexity parameterized by treewidth (or its path-like analogue pathwidth) has been determined under the so-called Strong Exponential Time Hypothesis (e.g., [33, 24, 32, 11, 28, 12, 16]). However, the main drawback of treewidth is that it is only bounded in sparse graphs: a graph on  $n$  vertices of treewidth  $k$  has no more than  $nk$  edges.

To capture the structure of dense graphs, several parameters have been introduced and considered. One of the most studied is clique-width. The clique-width of a graph is at most  $k$  if it can be constructed using the following four operations on  $k$ -labeled graphs: create a vertex with some label from  $1, \dots, k$ ; form a disjoint union of two already constructed graphs; give all vertices with label  $i$  label  $j$  instead; or create all edges between vertices



© Vera Chekan and Stefan Kratsch;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 35; pp. 35:1–35:15

Leibniz International Proceedings in Informatics



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

with labels  $i$  and  $j$ . It is known that if a graph has treewidth  $k$ , then it has clique-width at most  $3 \cdot 2^{k-1}$  and it is also known that an exponential dependence in this bound is necessary [9]. Conversely, cliques have clique-width at most 2 and unbounded treewidth. So on the one hand, clique-width is strictly more expressive than treewidth in the sense that if we can solve a problem efficiently on classes of graphs of bounded clique-width, then this is also true for classes of graphs of bounded treewidth. On the other hand, the exponential gap has the effect that as the price of solving the problem for larger graph classes we potentially obtain worse running times for some graph families.

Fürer introduced and studied two natural generalizations of clique-width, namely fusion-width (fw) [19] and multi-clique-width (mcw) [20]. For fusion-width, additionally to the clique-width operations, he allows an operator that fuses (i.e., merges) all vertices of label  $i$ . Originally, fusion-width (under a different name) was introduced by Courcelle and Makowsky [10]. However, they did not suggested studying it as a new width parameter since it is parametrically (i.e., up to some function) equivalent to clique-width. For multi-clique-width, the operations remain roughly the same as for clique-width but now every vertex is allowed to have multiple labels. For these parameters, Fürer showed the following relations to clique-width (cw) and treewidth (tw):

$$\text{fw} \leq \text{cw} \leq \text{fw} \cdot 2^{\text{fw}} \quad \text{mcw} \leq \text{cw} \leq 2^{\text{mcw}} \quad \text{fw} \leq \text{tw} + 2 \quad \text{mcw} \leq \text{tw} + 2 \quad (1)$$

Fürer also observed that the exponential gaps between clique-width and both fusion- and multi-clique-width are necessary. As our first result, we determine the relation between fusion-width and multi-clique-width:

► **Theorem 1.** *For every graph  $G$ , it holds that  $\text{mcw}(G) \leq \text{fw}(G) + 1$ . Moreover, given a fuse- $k$ -expression  $\phi$  of  $G$ , a multi-clique-width- $(k + 1)$ -expression of  $G$  can be created in time polynomial in  $|\phi|$  and  $k$ .*

The relations in (1) imply that a problem is FPT parameterized by fusion-width resp. multi-clique-width if and only if this is the case for clique-width. However, the running times of such algorithms might strongly differ. Fürer initiated a fine-grained study of problem complexities relative to multi-clique-width, starting with the INDEPENDENT SET problem. He showed that this problem can be solved in  $\mathcal{O}^*(2^{\text{mcw}})$  where  $\mathcal{O}^*$  hides factors polynomial in the input size. On the other hand, Lokshtanov et al. proved that under SETH no algorithm can solve this problem in  $\mathcal{O}^*((2 - \varepsilon)^{\text{pw}})$  where pw denotes the parameter called pathwidth [28]. Clique-width of a graph is at most its pathwidth plus two [15] so the same lower bound holds for clique-width and hence, multi-clique-width as well. Therefore, the tight dependence on both clique-width and multi-clique-width is the same, namely  $\mathcal{O}^*(2^k)$ . We show that this is the case for many further problems.

► **Theorem 2.** *Let  $G$  be a graph given together with a multi- $k$ -expression of  $G$ . Then:*

- *DOMINATING SET can be solved in time  $\mathcal{O}^*(4^k)$ ;*
- *$q$ -COLORING can be solved in time  $\mathcal{O}^*((2^q - 2)^k)$ ;*
- *CONNECTED VERTEX COVER can be solved in time  $\mathcal{O}^*(6^k)$ ;*
- *CONNECTED DOMINATING SET can be solved in time  $\mathcal{O}^*(5^k)$ .*

*And these results are tight under SETH.*

*Further, CHROMATIC NUMBER can be solved in time  $f(k) \cdot n^{2^{\mathcal{O}(k)}}$  and this is tight under ETH.*

We prove this by providing algorithms for multi-clique-width with the same running time as the known tight algorithms for clique-width. The lower bounds for clique-width known from the literature then apply to multi-clique-width as well proving the tightness of our results.

By Theorem 1, these results also apply to fusion-width. For the following three problems we obtain similar tight bounds relative to fusion-width as for clique-width, but it remains open whether the same is true relative to multi-clique-width:

► **Theorem 3.** *Let  $G$  be a graph given together with a fuse- $k$ -expression of  $G$ . Then:*

- *MAX CUT can be solved in time  $f(k) \cdot n^{\mathcal{O}(k)}$ ;*
- *EDGE DOMINATING SET can be solved in time  $f(k) \cdot n^{\mathcal{O}(k)}$ ;*
- *HAMILTONIAN CYCLE can be solved in time  $f(k) \cdot n^{\mathcal{O}(k)}$ .*

*And these results are tight under ETH.*

To prove these upper bounds, we provide an alternative view on fuse-expressions, called *glue-expressions*, interesting on its own. We show that a fuse- $k$ -expression can be transformed into a glue- $k$ -expression in polynomial time and then present dynamic-programming algorithms on glue-expressions. Due to the exponential gap between clique-width and both fusion- and multi-clique-width, our results provide exponentially faster algorithms on graphs witnessing these gaps.

**Related Work.** Two parameters related to both treewidth and clique-width are modular treewidth (mtw) [3, 22] and twinclass-treewidth [29, 31, 27] (unfortunately, sometimes also referred to as modular treewidth). It is known that  $\text{mcw} \leq \text{mtw} + 3$  (personal communication with Falko Hegerfeld). Further dense parameters have been widely studied in the literature. Rank-width (rw) was introduced by Oum and Seymour and it reflects the  $\mathbb{F}_2$ -rank of the adjacency matrices in the so-called branch decompositions. Originally, it was defined to obtain a fixed-parameter approximation of clique-width [30] by showing that  $\text{rw} \leq \text{cw} \leq 2^{\text{rw} + 1} - 1$ . Later, Bui-Xuan et al. started the study of algorithmic properties of rank-width [5]. Recently, Bergougnoux et al. proved the tightness of first ETH-tight lower bounds for this parameterization [2]. Another parameter defined via branch-decompositions and reflecting the number of different neighborhoods across certain cuts is boolean-width (boolw), introduced by Bui-Xuan et al. [6, 7]. Fürer [20] showed that  $\text{boolw} \leq \text{mcw} \leq 2^{\text{boolw}}$ . Recently, Eiben et al. presented a framework unifying the definitions and algorithms for computation of many graph parameters [13].

**Organization.** We start with some required definitions and notations in Section 2. In Section 3 we prove the relation between fusion-width and multi-clique-width from Theorem 1. After that, in Section 4 we introduce glue- $k$ -expressions and show how to obtain such an expression given a fuse- $k$ -expression of a graph. Then in Section 5 we employ these expressions to obtain algorithms parameterized by fusion-width. In Section 6 we present algorithms parameterized by multi-clique-width. We conclude with some open questions in Section 7. In this work some technical details have been omitted due to space constraints. We refer to the full version of the paper for all proofs [8].

## 2 Preliminaries

For  $k \in \mathbb{N}_0$ , we denote by  $[k]$  the set  $\{1, \dots, k\}$  and we denote by  $[k]_0$  the set  $[k] \cup \{0\}$ .

We use standard graph-theoretic notation. Our graphs are simple and undirected if not explicitly stated otherwise. For a graph  $H$  and a partition  $(V_1, V_2)$  of  $V(H)$ , by  $E_H(V_1, V_2) = \{\{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2\}$  we denote the set of edges between  $V_1$  and  $V_2$ . For a set  $S$  of edges in a graph  $H$ , by  $V(S)$  we denote the set of vertices incident with the edges in  $S$ .

A  $k$ -labeled graph is a pair  $(H, \text{lab}_H)$  where  $\text{lab}_H: V(H) \rightarrow [k]$  is a *labeling function* of  $H$ . Sometimes to simplify the notation in our proofs we will allow the labeling function to map to some set of cardinality  $k$  instead of the set  $[k]$ . In the following, if the number  $k$  of labels does not matter, or it is clear from the context, we omit  $k$  from the notions (e.g., a labeled graph instead of a  $k$ -labeled graph). Also, if the labeling function is clear from the context, then we simply call  $H$  a labeled graph as well. Also we sometimes omit the subscript  $H$  of the labeling function  $\text{lab}_H$  for simplicity. For  $i \in [k]$ , by  $U_i^H = \text{lab}_H^{-1}(i)$  we denote the set of vertices of  $H$  with label  $i$ . We consider the following four operations on  $k$ -labeled graphs.

1. *Introduce*: For  $i \in [k]$ , the operator  $v\langle i \rangle$  creates a graph with a single vertex  $v$  that has label  $i$ . We call  $v$  the *title* of the vertex.
2. *Union*: The operator  $\oplus$  takes two vertex-disjoint  $k$ -labeled graphs and creates their disjoint union. The labels are preserved.
3. *Join*: For  $i \neq j \in [k]$ , the operator  $\eta_{i,j}$  takes a  $k$ -labeled graph  $H$  and creates the supergraph  $H'$  on the same vertex set with  $E(H') = E(H) \cup \{\{u, v\} \mid \text{lab}_H(u) = i, \text{lab}_H(v) = j\}$ . The labels are preserved.
4. *Relabel*: For  $i \neq j$ , the operator  $\rho_{i \rightarrow j}$  takes a  $k$ -labeled graph  $H$  and creates the same  $k$ -labeled graph  $H'$  apart from the fact that every vertex that with label  $i$  in  $H$  instead has label  $j$  in  $H'$ .

A well-formed sequence of such operations is called a  $k$ -*expression* or a *clique-expression*. With a  $k$ -expression  $\phi$  one can associate a rooted tree such that every node corresponds to an operator, this tree is called a *parse tree* of  $\phi$ . With a slight abuse of notation, we denote it by  $\phi$  as well. By  $G^\phi$  we denote the labeled graph arising in  $\phi$ . And for a node  $t$  of  $\phi$  by  $G_t^\phi$  we denote the labeled graph arising in the subtree (sometimes also called a *sub-expression*) rooted at  $t$ , this subtree is denoted by  $\phi_t$ . The graph  $G_t^\phi$  is then a subgraph of  $G^\phi$ . A graph  $H$  has *clique-width* of at most  $k$  if there is a labeling function  $\text{lab}_H$  of  $H$  and a  $k$ -expression  $\phi$  such that  $G^\phi$  is equal to  $(H, \text{lab}_H)$ . By  $\text{cw}(H)$  we denote the smallest integer  $k$  such that  $H$  has clique-width at most  $k$ . Fürer has studied two generalizations of  $k$ -expressions [19, 20].

*Fuse*: For  $i \in [k]$ , the operator  $\theta_i$  takes a  $k$ -labeled graph  $H$  with  $\text{lab}_H^{-1}(i) \neq \emptyset$  and *fuses* the vertices with label  $i$ , i.e., the arising graph  $H'$  has vertex set  $(V(H) - \text{lab}_H^{-1}(i)) \cup \{v\}$ , the edge relation in  $V(H) - \text{lab}_H^{-1}(i)$  is preserved, and  $N_{H'}(v) = N_H(\text{lab}_H^{-1}(i))$ . The labels of vertices in  $V(H') - v$  are preserved, and vertex  $v$  has label  $i$ . A *fuse- $k$ -expression* is a well-formed expression that additionally to the above four operations is allowed to use fuses. We adopt the above notations from  $k$ -expressions to fuse- $k$ -expressions. Let us only remark that for a node  $t$  of a fuse- $k$ -expression  $\phi$ , the graph  $G_t^\phi$  is not necessarily a subgraph of  $G^\phi$  since some vertices of  $G_t^\phi$  might be fused later in  $\phi$ .

► **Remark 4.** Originally, Fürer allows that a single introduce-node creates multiple, say  $q$ , vertices with the same label. However, we can eliminate such operations from a fuse-expression  $\phi$  as follows. If the vertices introduced at some node participate in some fuse later in the expression, then it suffices to introduce only one of them. Otherwise, we can replace this introduce-node by  $q$  nodes introducing single vertices combined using union-nodes. These vertices are then also the vertices of  $G^\phi$ . So in total, replacing all such introduce-nodes would increase the number of nodes of the parse tree by at most  $\mathcal{O}(|V(G^\phi)|)$ , which is not a problem for our algorithmic applications.

Another generalization of clique-width introduced by Fürer [20] is multi-clique-width (mcw). A multi- $k$ -labeled graph is a pair  $(H, \text{lab}_H)$  where  $\text{lab}_H: V(H) \rightarrow 2^{[k]}$  is a multi-labeling function. We consider the following four operations of multi- $k$ -labeled graphs.

1. *Introduce*: For  $q \in [k]$  and  $i_1, \dots, i_q \in [k]$ , the operator  $v\langle i_1, \dots, i_q \rangle$  creates a multi- $k$ -labeled graph with a single vertex that has label set  $\{i_1, \dots, i_q\}$ .
2. *Union*: The operator  $\oplus$  takes two vertex-disjoint multi- $k$ -labeled graphs and creates their disjoint union. The labels are preserved.
3. *Join*: For  $i \neq j \in [k]$ , the operator  $\eta_{i,j}$  takes a multi- $k$ -labeled graph  $H$  and creates its supergraph  $H'$  on the same vertex set with  $E(H') = E(H) \cup \{\{u, v\} \mid i \in \text{lab}_H(u), j \in \text{lab}_H(v)\}$ . This operation is only allowed when there is no vertex in  $H$  with labels  $i$  and  $j$  simultaneously, i.e., for every vertex  $v$  of  $H$  we have  $\{i, j\} \not\subseteq \text{lab}_H(v)$ . The labels are preserved.
4. *Relabel*: For  $i \in [k]$  and  $S \subseteq [k]$ , the operator  $\rho_{i \rightarrow S}$  takes a multi- $k$ -labeled graph  $H$  and creates the same multi-labeled graph apart from the fact that every vertex with label set  $L \subseteq [k]$  such that  $i \in L$  in  $H$  instead has label set  $(L \setminus \{i\}) \cup S$  in  $H'$ . Note that  $S = \emptyset$  is allowed.

A well-formed sequence of these four operations is called a *multi- $k$ -expression*. As for fuse-expressions, Fürer allows introduce-nodes to create multiple vertices but we can eliminate this by increasing the number of nodes in the expression by at most  $\mathcal{O}(|V(G^\phi)|)$ . We adopt the analogous notations from  $k$ -expressions to multi- $k$ -expressions.

**Complexity.** To the best of our knowledge, the only known way to approximate multi-clique-width and fusion-width is via clique-width, i.e., to employ the relation (1). The only known way to approximate clique-width is, in turn, via rank-width. This way we obtain a  $2^{2^k}$ -approximation of multi-clique-width and fusion-width running in FPT time. For this reason, to obtain tight running times in our algorithms we always assume that a fuse- or multi- $k$ -expression is provided. Let us emphasize that this is also the case for all tight results for clique-width in the literature (see e.g., [1, 27]). In this work, we will show that if a graph admits a multi- $k$ -expression resp. a fuse- $k$ -expression, then it also admits one whose size is polynomial in the size of the graph. Moreover, such a “compression” can be carried out in time polynomial in the size of the original expression. Therefore, we delegate this compression to a black-box algorithm computing or approximating multi-clique-width or fusion-width and assume that provided expressions have size polynomial in the graph size.

**(Strong) Exponential Time Hypothesis.** The algorithms in this work are tight under one of the following conjectures formulated by Impagliazzo et al. [23]. The Exponential Time Hypothesis (ETH) states that there is  $0 < \varepsilon < 1$  such that 3-SAT with  $n$  variables and  $m$  clauses cannot be solved in time  $\mathcal{O}^*(2^{\varepsilon n})$ . The Strong Exponential Time Hypothesis (SETH) states that for every  $0 < \varepsilon < 1$  there is an integer  $q$  such that  $q$ -SAT cannot be solved in time  $\mathcal{O}^*(2^{\varepsilon n})$ . In this work,  $\mathcal{O}^*$  hides factors polynomial in the input size.

**Simplifications.** If the graph is clear from the context, by  $n$  we denote the number of its vertices. If not stated otherwise, the number of labels is denoted by  $k$  and a label is a number from  $[k]$ .

### 3 Relation Between Fusion-Width and Multi-Clique-Width

In this section, we show that the multi-clique-width of a graph is at most as large as its fusion-width plus one. Fürer [20] has proven the following relation:

► **Theorem 5** ([20]). *For every graph  $H$ , it holds that  $\text{cw}(H) \leq \text{fw}(H) + 1$ .*

The proof is constructive: given a fuse- $k$ -expression it creates a  $k \cdot 2^k$ -expression of the same graph. We use ideas from this proof to prove our result.

► **Theorem 6.** *For every graph  $H$ , it holds that  $\text{mcw}(H) \leq \text{fw}(H) + 1$ . Moreover, given a fuse- $k$ -expression  $\phi$  of  $H$ , a multi- $(k+1)$ -expression of  $H$  can be created in time polynomial in  $|\phi|$  and  $k$ .*

**Proof.** Here we sketch the intuition behind the proof and for all formalities we refer to the full version. We start by showing that  $\text{mcw}(H) \leq 2 \cdot \text{fw}(H)$  holds. To prove this, we will consider a fuse- $k$ -expression of  $H$  and from it, we will construct a multi- $2k$ -expression of  $H$  using labels  $\{1, \dots, k, \widehat{1}, \dots, \widehat{k}\}$ . For simplicity, let  $\widehat{[k]} = \{\widehat{1}, \dots, \widehat{k}\}$ . For this first step, we follow the construction of Fürer in his proof of Theorem 5. There he uses  $k \cdot 2^k$  labels from the set  $[k] \times 2^{[k]}$  so the second component of such a label is a subset of  $[k]$ . Multi-expressions allow vertices to hold multiple labels and we model the second component of a label via subsets of  $\widehat{[k]}$ . After that, we show that the labels  $i$  and  $\widehat{i}$  can be almost unified for every  $i \in [k]$ . Using just one additional label  $\star$ , we then obtain a multi- $(k+1)$ -expression of  $H$ .

For simplicity of representation, in this proof sketch we assume that our fuse-expression contains no relabel-nodes (we refer to the full version for the complete proof). We may assume that our fuse-expression does not contain nodes that do not change the arising labeled graph. Also if a vertex arising in some fuse-operation participates in some later fuse-operation, then the earlier fuse can be removed. Now we assume that any vertex arising in a fuse-operation does not participate in later fuses. We say that  $v$  is a *fuse-vertex* at a node  $x$  of the expression if  $v$  *participates* in some fuse-operation above  $x$ . For the label  $i$  of  $v$  we then also say that  $i$  is a *fuse-label* at  $x$ . Instead of first creating the fuse-vertices via introduce-nodes and then fusing them, we will introduce only one vertex representing the result of the fusion. The creation of the edges incident with such a *new* vertex (originally incident with fuse-vertices) then needs to be *postponed* until the moment where this vertex is introduced. To remember that some vertex  $v$  is missing an edge to a new vertex with label  $i$ , we will add a label  $\widehat{i}$  to the label set of  $v$ . After the creation of this vertex, the edge will be reconstructed using a corresponding join.

Now we provide more details. First, we cut off every leaf of the expression introducing a fuse-vertex. Second, we replace every fuse-node  $\theta_i$  by a *new* introduce-node  $1\langle i \rangle$ . Since for every fuse-vertex we have kept only the latest fuse-node it participates in, the vertex set of the graph arising in the current expression is  $V(H)$ , and the edges incident with new vertices need to be created. For this, let  $x$  be a  $\eta_{i,j}$ -node. If both  $i$  and  $j$  are not fuse-labels at  $x$ , no additional work needs to be done. Next, assume that exactly one of the labels  $i$  and  $j$ , say  $i$ , is a fuse-label at  $x$ . The information about the created edges is stored in vertices of label  $j$ : for this, we replace this join with a relabel  $\rho_{j \rightarrow \{j, \widehat{i}\}}$ . Now assume that both  $i$  and  $j$  are fuse-labels at  $x$ . Then  $x$  creates only one edge of  $H$  since all vertices of label  $i$  (resp.  $j$ ) are fused into one vertex later. We may assume that in the original expression, the fuse of vertices with label  $j$  happens before the fuse of label  $i$ . We store the information about the postponed edge in  $j$  as follows. Let  $x_j$  be the new introduce-node that replaced the fuse-node for label  $j$ . And let  $S$  denote the set of labels of the new vertex created in this node: in the beginning,  $S$  consists of  $j$  only but after processing some join-nodes, it might contain further labels from  $\widehat{[k]}$ . Then we replace  $x_j$  with a  $1\langle S \cup \{\widehat{i}\} \rangle$ -node. Finally, we create the remembered edges as follows. Let  $x$  be a new  $1\langle S \rangle$ -node for some set  $S$  of labels. By construction, there exists a unique  $i \in S \cap [k]$ : this is the label of vertices originally fused at this node while  $S \setminus i \subseteq \widehat{[k]}$  remembers the edges to be created later in the expression. So right after  $x$ , we first add a  $\eta_{i, \widehat{i}}$ -node and second, a  $\rho_{i \rightarrow \emptyset}$ -node to reflect that the missing edges have been created.

This concludes the construction of a multi- $2k$ -expression of  $H$ . The crucial observation is that the labels  $i$  and  $\widehat{i}$  are almost never used at the same time. Indeed, if label  $\widehat{i}$  occurs in some vertex created by some sub-expression, then all leaves introducing the vertices of label  $i$  have been cut off from this sub-expression. So there are no vertices with label  $i$  now. The only moment to which both labels  $i$  and  $\widehat{i}$  occur simultaneously is right after a new  $1\langle S \rangle$ -node with  $i \in S$ : now the label  $\widehat{i}$  is used to create the postponed edges incident with the new vertex and then  $\widehat{i}$  is removed. So apart from these two operations, we can unify  $i$  and  $\widehat{i}$ : one additional label  $\star$  suffices to distinguish them during these operations. This results in a multi- $(k+1)$ -expression of  $H$ . ◀

#### 4 Reduced Glue-Expressions

► **Definition 7.** A glue- $k$ -expression is a well-formed expression constructed from introduce-, join-, relabel-, and glue-operations on  $k$ -labeled graphs. A glue-operation takes as input two  $k$ -labeled graphs  $(H_1, \text{lab}_1)$  and  $(H_2, \text{lab}_2)$  satisfying the following two properties:

- For every  $v \in V(H_1) \cap V(H_2)$ , the vertex  $v$  has the same label in  $H_1$  and  $H_2$ , i.e., we have  $\text{lab}_1(v) = \text{lab}_2(v)$ .
- For every  $v \in V(H_1) \cap V(H_2)$  and every  $j \in [2]$ , the vertex  $v$  is the unique vertex with its label in  $H_j$ , i.e., we have  $|\text{lab}_1^{-1}(\text{lab}_1(v))| = |\text{lab}_2^{-1}(\text{lab}_2(v))| = 1$ .

In this case, we call the graphs  $H_1$  and  $H_2$  glueable. The output of this operation is then the graph  $H_1 \sqcup H_2$  with  $V(H_1 \sqcup H_2) = V(H_1) \cup V(H_2)$  and  $E(H_1 \sqcup H_2) = E(H_1) \cup E(H_2)$  where the labels are preserved. The vertices in  $V(H_1) \cap V(H_2)$  are called glue-vertices.

Note that if  $H_1$  and  $H_2$  satisfy these properties, then the gluing is equivalent to a union followed by a sequence of fuses  $\theta_i$  where  $i$  is a label of a vertex shared by  $H_1$  and  $H_2$ .

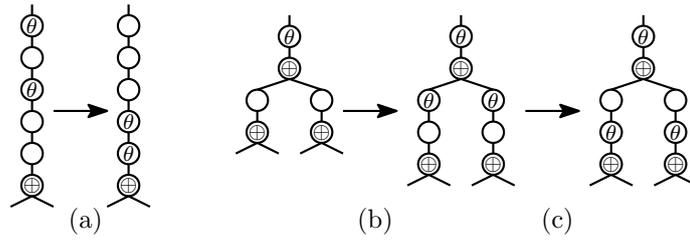
► **Definition 8.** A glue- $k$ -expression  $\xi$  is called reduced if the following properties hold:

1. Let  $i, j \in [k]$ , let  $t$  be a  $\eta_{i,j}$ -node in  $\xi$ , and let  $t'$  be the child of  $t$  in  $\xi$ . Then  $G_v^\xi$  contains no edge  $\{v, w\}$  with  $\text{lab}_v^\xi(v) = i$  and  $\text{lab}_v^\xi(w) = j$ .
2. Let  $t$  be a glue-node in  $\xi$  and let  $t_1$  and  $t_2$  be its children. Then the graphs  $G_{t_1}^\xi$  and  $G_{t_2}^\xi$  are edge-disjoint.
3. Let  $t$  be a glue-node in  $\xi$ , let  $t_1$  and  $t_2$  be its children, and let  $v$  be a glue-vertex. Then for every  $j \in [2]$ , the vertex  $v$  has an incident edge in  $G_{t_j}^\xi$ .

As we will see in Section 5, such expressions are very useful for algorithmic applications.

In this section we sketch how to transform a fuse- $k$ -expression into a reduced glue- $k$ -expression of the same graph in polynomial time. Although the idea behind the construction of such expressions is quite natural, the realization is rather technical. For this reason, here we only provide a high-level idea of the required steps and refer to the full version for the extensive description.

In the first phase, we transform a fuse- $k$ -expression into a (not necessarily reduced) glue- $k$ -expression as follows. As the first step, we want to achieve that every fuse-node  $t$  is right above some union-node, i.e., there are only fuse-nodes on the path between  $t$  and the topmost union-node below  $t$ . For this, we shift every fuse-node down to the closest union-node (see Figure 1 (a)). We emphasize that this cannot be achieved simply by repetitive swapping of the fuse-node with its child: e.g., swapping  $\theta_i$  with  $\rho_{j \rightarrow i}$  would change the arising graph. In the second step, we want to achieve that every fuse-node  $t$  fuses exactly two vertices and these vertices come from different sides of the union right below it. In other words, we want that any two vertices, that are ever fused, are fused as early as possible. So if there is a node  $t$  fusing at least two vertices coming from the same side of the topmost union-node  $t_\oplus$



■ **Figure 1** The steps to transform a fuse-expression into a glue-expression. (a) Shifting fuse-nodes to union-nodes. (b) Creating copies of fuse-nodes to fuse vertices as early as possible. (c) Shifting the copied fuse-nodes to union-nodes.

below it, these vertices can actually be fused before  $t_{\oplus}$  already. To accomplish this, we add a new fuse-node  $t'$  below  $t_{\oplus}$  (see Figure 1 (b)). After that,  $t'$  is shifted down to the next union-node as in the first step (see Figure 1 (c)). In this second step, one needs to be careful in order to ensure that the process terminates at all and that it takes only polynomial time. For this, we choose an appropriate order for processing the fuse-nodes. After these two steps, we replace every sequence consisting of a union-node and following fuse-nodes with a glue-node to obtain a glue-expression.

In the second phase, we make our glue-expression reduced as follows (see Definition 8). First, given two join-nodes creating the same edge, we can show that at least of them can be safely removed. Second, if a glue-vertex of two glued graphs has no incident edge in one of them, then we can remove this vertex from that graph without changing the result of the gluing. We refer to the full version for details. Altogether, we prove the following:

► **Theorem 9.** *Let  $\phi$  be a fuse- $k$ -expression of a graph  $H$  on  $n$  vertices and  $m$  edges. Then in time polynomial in  $|\phi|$  and  $k$  we can compute a reduced glue- $k$ -expression  $\zeta$  of  $H$  such that the parse tree of  $\zeta$  contains  $\mathcal{O}(k^2(m+n))$  nodes.*

Let us remark, that unlike clique-expressions (whose leaves are in bijection with the vertices of the arising graph), the number of leaves in a fuse-expression can be unbounded in general. This is due to fuse-nodes reducing the number of vertices in the constructed graph. So Theorem 9 in particular shows that a polynomial number of introduce-nodes suffices.

## 5 Algorithms Parameterized by Fusion-Width

In this section we show how to employ reduced glue-expressions to obtain algorithms parameterized by fusion-width (given a fuse-expression of corresponding width). In the previous section, we showed that every fuse- $k$ -expression can be transformed into a reduced glue- $k$ -expression of small size in polynomial time. So for the remainder of this section we assume that a glue- $k$ -expression  $\phi$  of an input graph  $G$  is given. Recall that in particular, two graphs glued at any glue-node of  $\phi$  are edge-disjoint.

All algorithms in this section have running time  $f(\text{fw})n^{\mathcal{O}(\text{fw})}$ . For each of the problems considered here, in the literature there is a lower bound stating that under ETH, the problem cannot be solved in time  $f(\text{cw})n^{\mathcal{O}(\text{cw})}$  even if a clique-expression of corresponding width is provided [17, 18]. Since fusion-width is at most as large as clique-width, these lower bounds hold for fusion-width as well implying the tightness of our results. Fürer observed that there exist graphs whose clique-width is exponential in their fusion-width [19]. Therefore, in addition to solving these problems for a larger class of graphs, we obtain an exponential improvement in the running time for some families of graphs. The only difference between glue- and clique-expressions is the possibility of using glue-nodes so it will suffice to extend existing dynamic-programming algorithms from the literature to glue-nodes.

## 5.1 Max Cut

In this problem, given a graph  $G = (V, E)$  we are asked about the maximum cardinality of  $E_G(V_1, V_2)$  over all partitions  $(V_1, V_2)$  of  $V$ . Fomin et al. have developed an  $n^{\mathcal{O}(\text{cw})}$  algorithm [17], which we now extend to reduced glue-expressions by showing how to deal with glue-nodes. Before this, a small remark: the correctness of their algorithm for a join-node  $t$  requires that none of edges created by  $t$  is already present in the graph. A reduced glue-expression satisfies this property so the procedures for all node types other than glue-nodes can indeed be adopted.

Their dynamic-programming tables are defined as follows. For a graph  $H$ , the table  $T_H$  contains all vectors  $h = (s_1, \dots, s_k, r)$  with  $0 \leq s_i \leq |U_i^H|$  for every  $i \in [k]$  and  $0 \leq r \leq |E(H)|$  for which there exists a partition  $(V_1, V_2)$  of  $V(H)$  such that  $|V_1 \cap U_i^H| = s_i$  for every  $i \in [k]$  and there are at least  $r$  edges between  $V_1$  and  $V_2$  in  $H$ . We say that the partition  $(V_1, V_2)$  *witnesses* the vector  $h$ . Then the output of the algorithm is the largest integer  $r$  such that  $T_G$  contains an entry  $(s_1, \dots, s_k, r)$  for some  $s_1, \dots, s_k \in \mathbb{N}_0$ . Processing a fuse-operation  $\theta_i$  applied to some subgraph  $H$  seems to be problematic in this setting for the following reason. Some vertices of label  $i$  might have common neighbors so after the application of the fuse-operator, multiple edges fall together. Given the table  $T_H$  only we cannot deduce how many of those edges were running across the partition and it is unclear how to update the table correctly. To avoid such issues, we replace fuse-nodes with glue-nodes.

Now we provide a way to compute the table  $T_H$  if  $H = H^1 \sqcup H^2$  for two glueable edge-disjoint  $k$ -labeled graphs  $H^1$  and  $H^2$  if the tables  $T_{H^1}$  and  $T_{H^2}$  are provided. Let  $\{v_1, \dots, v_q\} = V(H^1) \cap V(H^2)$  for some  $q \in \mathbb{N}_0$  and let  $i_1, \dots, i_q$  be the labels of  $v_1, \dots, v_q$  in  $H^1$ , respectively. Glueability implies that for every  $j \in [q]$ , it holds that  $|U_{i_j}^{H^1}| = |U_{i_j}^{H^2}| = 1$ . Hence, for every entry  $(s_1, \dots, s_k, r)$  of  $T_{H^1}$  and every  $j \in [q]$ , it also holds that  $s_{i_j} \in \{0, 1\}$  with  $s_{i_j} = 1$  if and only if  $v_j$  is put into  $V_1$  in the partition witnessing this entry. The same holds for the entries in  $T_{H^2}$ . This gives the following way to compute the table  $T_H$ . We initialize this table to be empty. Then we iterate through all pairs of vectors  $h^1 = (s_1^1, \dots, s_k^1, r^1)$  from  $T_{H^1}$  and  $h^2 = (s_1^2, \dots, s_k^2, r^2)$  from  $T_{H^2}$ . If there is an index  $j \in [q]$  such that  $s_{i_j}^1 \neq s_{i_j}^2$ , then we skip this pair. Otherwise, for every  $0 \leq r \leq r^1 + r^2$ , we add to  $T_H$  the vector  $h = (s_1, \dots, s_k, r)$  where for all  $i \in [k]$

$$s_i = \begin{cases} s_i^1 + s_i^2 & i \notin \{i_1, \dots, i_q\} \\ s_i^1 \cdot s_i^2 & i \in \{i_1, \dots, i_q\} \end{cases}.$$

Note that for  $i \in \{i_1, \dots, i_q\}$ , the above definition simply states that we have  $s_i = 1$  iff both  $s_i^1$  and  $s_i^2$  are equal to 1, and  $s_i = 0$  iff both  $s_i^1$  and  $s_i^2$  are equal to 0. It is not difficult to verify the correctness of this procedure. If there are no glue-vertices, then our approach coincides with the one for union-nodes by Fomin et al. If there is a glue-vertex, say  $v$ , then we consider partitions  $(V_1^1, V_2^1)$  and  $(V_1^2, V_2^2)$  of  $H^1$  and  $H^2$ , respectively, putting  $v$  on the same side. Then a partition of  $H$  naturally arises as a union  $(V_1^1 \cup V_1^2, V_1^1 \cup V_2^2)$ . Since the graphs are edge-disjoint, every edge crossing this partition crosses exactly one of the partitions  $(V_1^1, V_2^1)$  and  $(V_1^2, V_2^2)$  implying that the choice of  $r$  is correct. We refer to the full version for a formal proof.

Recall that every graph constructed from any sub-expression of  $\phi$  is a subgraph of  $G$  so the tables for graphs  $H^1$  and  $H^2$  contain  $n^{\mathcal{O}(k)}$  entries. Thus, the procedure for glue-nodes runs in time  $n^{\mathcal{O}(k)}$ . By Theorem 9 we may assume that  $\phi$  contains a polynomial number of nodes. Altogether, we obtain an algorithm solving MAX CUT in time  $n^{\mathcal{O}(\text{fw})}$ . Fomin et al. have also proven that under ETH, it cannot be solved in time  $f(\text{cw})n^{\mathcal{O}(\text{cw})}$  for any computable function  $f$  (see [17] Theorem 4.1) so our result is tight.

## 5.2 Edge Dominating Set

In this problem, given a graph  $G = (V, E)$  we are asked about the cardinality of a minimum set  $X \subseteq E$  such that every edge in  $E$  either belongs to  $X$  itself or it has an incident edge in  $X$ . Fomin et al. have developed an  $n^{\mathcal{O}(\text{cw})}$  algorithm solving this problem [17], which we now extend to reduced glue-expressions. As for MAX CUT, the algorithm requires that for any join-node, the edges created by this node are not yet present in the graph. A reduced glue-expression satisfies this property so their procedures for introduce-, join-, and relabel-nodes can be adopted.

For a  $k$ -labeled graph  $H$ , the table  $T_H$  contains all vectors  $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$  of non-negative integers such that there exists a set  $S \subseteq E(H)$  and a set  $R \subseteq V(H) \setminus V(S)$  with the following properties:

- $|S| \leq \ell \leq |E(H)|$ ;
- for every  $i \in [k]$ , exactly  $s_i$  vertices of  $U_i^H$  are incident with edges in  $S$ ;
- for every  $i \in [k]$ , we have  $|R \cap U_i^H| = r_i$ ;
- every edge of  $H$  undominated by  $S$  has an end-vertex in  $R$ .

We say that the pair  $(S, R)$  witnesses the vector  $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$  in  $H$ . The last property reflects that it is possible to attach a *pendant* edge to every vertex in  $R$  so that the set  $S$  together with these pendant edges dominates all edges of  $H$ . The size of the minimum edge dominating set of  $G$  is then the smallest integer  $\ell$  such that the table  $T_G$  contains an entry  $(s_1, \dots, s_k, 0, \dots, 0, \ell)$  for some  $s_1, \dots, s_k \in \mathbb{N}_0$ .

To complete the algorithm for the fusion-width parameterization, we provide a way to compute the table  $T_H$  if  $H = H^1 \sqcup H^2$  for two glueable edge-disjoint  $k$ -labeled graphs  $H^1$  and  $H^2$  if the tables  $T_{H^1}$  and  $T_{H^2}$  are provided. Let  $\{v_1, \dots, v_q\} = V(H^1) \cap V(H^2)$  for some  $q \in \mathbb{N}_0$  and let  $i_1, \dots, i_q$  be the labels of  $v_1, \dots, v_q$  in  $H^1$ , respectively. Then for every  $j \in [q]$ , it holds that  $|U_{i_j}^{H^1}| = |U_{i_j}^{H^2}| = 1$ . Hence, for every entry  $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$  of  $T_{H^1}$  and every  $j \in [q]$ , it holds that  $s_{i_j} + r_{i_j} \leq 1$ . The same holds for the entries in  $T_{H^2}$ . This motivates the following way to compute the table  $T_H$ . We initialize this table to be empty. Then we iterate through all pairs of vectors  $h^1 = (s_1^1, \dots, s_k^1, r_1^1, \dots, r_k^1, \ell^1)$  from  $T_{H^1}$  and  $h^2 = (s_1^2, \dots, s_k^2, r_1^2, \dots, r_k^2, \ell^2)$  from  $T_{H^2}$  and for every  $\ell^1 + \ell^2 \leq \ell \leq |E(H)|$ , we add to  $T_H$  the vector  $h = (s_1, \dots, s_k, r_1, \dots, r_k, \ell)$  defined as follows. For every  $i \in [k] \setminus \{i_1, \dots, i_q\}$ , it holds that  $s_i = s_i^1 + s_i^2$  and  $r_i = r_i^1 + r_i^2$ . And for every  $i \in \{i_1, \dots, i_q\}$ , it holds that  $s_i = s_i^1 \vee s_i^2$  and  $r_i = \neg s_i^1 \wedge \neg s_i^2 \wedge (r_i^1 \vee r_i^2)$ .

To argue the correctness, we sketch only one direction here. The other is similar and for all details refer to the full version. We now show that if  $h_1$  and  $h_2$  belong to  $T_{H^1}$  and  $T_{H^2}$ , respectively, then the vector  $h$  indeed belongs to  $T_H$ . So let  $(S^1, R^1)$  witness  $h^1$  in  $H^1$  and let  $(S^2, R^2)$  witness  $h^2$  in  $H^2$ . Then we set  $S = S^1 \cup S^2$  and construct  $R$  from  $R^1 \cup R^2$  by removing all vertices incident with  $S$ . Recall that every vertex in  $R^1$  has no incident edge in  $S^1$ . So a vertex  $v \in R^1 \setminus R$  must have an incident edge in  $S^2$  by construction and therefore, the vertex  $v$  must be a glue-vertex. The analogous is true for  $R^2 \setminus R$ . With this, one can verify that  $R$  complies with  $r_1, \dots, r_k$ . Also it is straight-forward to verify that this is true for  $S$  and  $s_1, \dots, s_k$ . The bound  $\ell^1 + \ell^2 \leq \ell$  implies that the size of  $S$  is at most  $\ell$  (when proving the other direction, we use that  $H^1$  and  $H^2$  are edge-disjoint). So it remains to show that every edge of  $H$  undominated by  $S$  has an end-point in  $R$ . Recall that  $E(H) = E(H^1) \cup E(H^2)$ . Let  $e$  be an edge of  $E(H^1)$  undominated by  $S = S^1 \cup S^2$ . Since it is not dominated by  $S^1$ , it has an end-point, say  $v$ , in  $R^1$ . By construction of  $R$ , either  $v$  still belongs to  $R$  or it has an incident edge in  $S^2$ . We have assumed that  $e$  is not dominated by  $S^1 \cup S^2$  so  $v$  belongs to  $R$  as desired. A symmetric argument applies to edges of  $H^2$ . This concludes the proof that  $h$  is an entry of  $T_H$ .

As in the previous subsection, for any subgraph  $H$  of  $G$  constructed in some sub-expression of  $\phi$ , the table  $T_H$  contains  $n^{\mathcal{O}(k)}$  entries and we obtain an algorithm solving EDGE DOMINATING SET in time  $n^{\mathcal{O}(fw)}$ . Fomin et al. have also proven that under ETH, it cannot be solved in time  $f(cw)n^{\mathcal{O}(cw)}$  for any computable function  $f$  (see [17] Theorem 5.1) so our result is tight.

### 5.3 Hamiltonian Cycle

In this problem, given a graph  $G = (V, E)$  we are asked about the existence of a cycle visiting each vertex exactly once. Our algorithm relies on the algorithm by Bergougnoux et al. [1] running in time  $f(cw)n^{\mathcal{O}(cw)}$ . A partial solution for this problem is usually a path packing, that is, a set of paths containing every vertex of the graph (constructed by the current sub-expression) exactly once. The earlier  $f(cw)n^{\mathcal{O}(cw^2)}$  algorithm by Espelage et al. stores for every pair of labels  $i$  and  $j$ , the number of paths in the path packing between a vertex with label  $i$  and a vertex with label  $j$  [14]. This naturally defines a graph on  $k$  vertices in which there is an edge for every path in the path packing. Bergougnoux et al. [1] show that instead of keeping track of all edges, it suffices to remember the degree sequence and the set of connected components of this graph. To obtain an algorithm relying on this idea they employ the technique of so-called representative sets: they define what does it mean for a set of partial solutions to be representative and then show that their procedures for nodes of a clique-expression maintain representativity.

To extend this algorithm to the parameterization by fusion-width, we describe how to handle glue-nodes. Let  $H_1$  and  $H_2$  be two edge-disjoint glueable graphs and let  $\mathcal{P}_1$  and  $\mathcal{P}_2$  be path packings of  $H_1$  and  $H_2$ , respectively. Then a natural combination of these partial solutions is the gluing  $\mathcal{P} := \mathcal{P}_1 \sqcup \mathcal{P}_2$  (a subgraph of  $H_1 \sqcup H_2$ ). Unlike a union-node (as handled in [1]), the subgraph  $\mathcal{P}$  is not necessarily a path packing: first, glue-vertices might have degree larger than 2 in  $\mathcal{P}$  and second, cycles might occur (e.g., if two paths with the same end-vertices are glued). We can show that if we iterate over all  $\mathcal{P}_1$  and  $\mathcal{P}_2$  and filter out combinations  $\mathcal{P}_1 \sqcup \mathcal{P}_2$  that are not path packings, then we obtain exactly the set of all partial solutions of  $H_1 \sqcup H_2$ . After that, we show that this approach maintains representativity. This part is very technical and we refer to the full version for all details. Let us remark that although we follow the idea by Bergougnoux et al. [1], our proof of correctness gets more involved than the one for union-nodes from their work: when gluing two graphs, the partial solutions are combined in a less trivial way than when forming a disjoint union of two graphs. The tightness is implied by [18].

## 6 Algorithms Parameterized by Multi-Clique-Width

In this section, we consider algorithms for problems parameterized by multi-clique-width. For all of these problems, SETH-tight (and for CHROMATIC NUMBER even an ETH-tight) algorithms for clique-width are known, we refer to [18, 25, 27, 21] for the corresponding lower bounds. We show that algorithms with the same running time exist relative to multi-clique-width. The clique-width lower bounds then transfer and imply the tightness of our results. As for fusion-width, Fürer observed that there exist graphs whose clique-width is exponential in their multi-clique-width [20]. So we obtain exponentially faster algorithms for some graph classes.

For our results, we rely on existing algorithms for the parameterization by clique-width and show that these (almost) do not use the fact that every vertex holds exactly one label: Some of the algorithms use clique-expressions with certain properties (e.g., so-called

*irredundancy*) though, so we need to either show that such a property holds for multi-clique-width expressions or provide an alternative way to keep the algorithm correct. Now we will sketch for each of the problems what changes need to be carried out and refer to the full version of the paper for all details. Using simple transformations we may assume that every introduce-node has form  $1\langle i \rangle$  and every relabel-node has form  $\rho_{i \rightarrow \emptyset}$  or  $\rho_{i \rightarrow \{i,j\}}$  for some  $i \neq j \in [k]$ . Also we may assume that a multi- $k$ -expression contains at most  $\mathcal{O}(k^2n)$  nodes.

For DOMINATING SET, the algorithm by Bodlaender et al. runs in time  $\mathcal{O}^*(4^{cw})$  [4] and never uses the fact that every vertex holds exactly one label. So using a straight-forward procedure to handle new relabel-nodes (of form  $\rho_{i \rightarrow \emptyset}$  and  $\rho_{i \rightarrow \{i,j\}}$ ) we obtain a  $\mathcal{O}^*(4^{mcw})$  algorithm for this problem.

The situation is similar for the CHROMATIC NUMBER algorithm by Kobler and Rotics [26]. The only minor thing one needs to handle is that the algorithm assumes that every color used by a graph coloring appears on some label: in a multi-expression, it might occur that all labels are removed from a vertex. To avoid this issue, we increase the number of labels in the expression by 1 and ensure that every vertex holds the new label at all times. Also we provide a procedure for the new relabel-nodes. Apart from that, the algorithm remains the same.

In his work, Fürer presents an algorithm for  $q$ -COLORING parameterized by multi-clique-width but it is not tight yet. For the  $q$ -COLORING problem, a naive  $\mathcal{O}^*((2^q)^{cw})$  algorithm tracks for every label the set of colors used on this label. Lampis improves this running time to  $\mathcal{O}^*((2^q - 2)^{cw})$  by observing that the empty set of colors can only be used by an empty label, while all colors can only occur if the label does not participate in any join later (such a label is called *dead*): otherwise, a monochromatic edge would occur [27]. Hegerfeld and Kratsch employ the same idea to obtain an  $\mathcal{O}^*(6^{cw})$  algorithm for CONNECTED VERTEX COVER [21]. To define *dead* labels, they rely on the existence of so-called *irredundant* clique-expression whose existence is unknown for multi-clique-width. We show that one can still make these two algorithms work for multi-clique-width by using a slightly different definition of *dead* and their counterpart, namely *active*, labels. Finally, for CONNECTED DOMINATING SET, to extend the  $\mathcal{O}^*(5^{cw})$  algorithm by Hegerfeld and Kratsch [21], there slightly more work is needed to adapt their inclusion-exclusion technique to a multi-label setting.

## 7 Conclusion

In this work, we studied two generalizations of clique-width, namely fusion-width and multi-clique-width, both introduced by Fürer [19, 20]. First, we showed that the fusion-width of a graph is an upper bound for its multi-clique-width. For the other direction, the best upper bound we are aware of is  $fw \leq 2^{mcw}$  and we leave open whether this is tight. By extending existing algorithms for clique-width, we have obtained tight algorithms parameterized by multi-clique-width for DOMINATING SET, CHROMATIC NUMBER,  $q$ -COLORING, CONNECTED VERTEX COVER, and CONNECTED DOMINATING SET. The running times are the same as for (S)ETH-optimal algorithms parameterized by clique-width.

For HAMILTONIAN CYCLE, MAXCUT, and EDGE DOMINATING SET, we were not able to achieve analogous results and these complexities remain open. Instead, we have introduced glue-expressions equivalent to fuse-expressions and then we employed them for these three problems to obtain tight algorithms parameterized by fusion-width with the same running times as ETH-optimal algorithms for clique-width.

Finally, in all algorithms we assume that a multi- $k$ -expression / fuse- $k$ -expression is provided. However, the complexity of computing these parameters is unknown. To the best of our knowledge, the best approximation would proceed via clique-width, have FPT running time, and a double-exponential approximation ratio.

---

## References

- 1 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An optimal XP algorithm for hamiltonian cycle on graphs of bounded clique-width. *Algorithmica*, 82(6):1654–1674, 2020. doi:10.1007/s00453-019-00663-9.
- 2 Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. Tight lower bounds for problems parameterized by rank-width. 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 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.STACS.2023.11.
- 3 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nord. J. Comput.*, 7(1):14–31, 2000.
- 4 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In Petr Hliněný and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2010. doi:10.1007/978-3-642-15155-2\_17.
- 5 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discret. Appl. Math.*, 158(7):809–819, 2010. doi:10.1016/j.dam.2009.09.009.
- 6 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.
- 7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.
- 8 Vera Chekan and Stefan Kratsch. Tight algorithmic applications of clique-width generalizations, 2023. Technical report. doi:10.48550/arXiv.2307.04628.
- 9 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.
- 10 Bruno Courcelle and Johann A. Makowsky. Fusion in relational structures and the verification of monadic second-order properties. *Math. Struct. Comput. Sci.*, 12(2):203–235, 2002. doi:10.1017/S0960129501003565.
- 11 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.
- 12 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.
- 13 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 63:1–63:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.63.
- 14 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2\_12.

- 15 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.
- 16 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22–25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.
- 17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshantov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.
- 18 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshantov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.
- 19 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics – 11th Latin American Symposium, Montevideo, Uruguay, March 31 – April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2014. doi:10.1007/978-3-642-54423-1\_7.
- 20 Martin Fürer. Multi-clique-width. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9–11, 2017, Berkeley, CA, USA*, volume 67 of *LIPICs*, pages 14:1–14:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ITCS.2017.14.
- 21 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *CoRR*, abs/2302.03627, 2023. doi:10.48550/arXiv.2302.03627.
- 22 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. *CoRR*, abs/2302.14128, 2023. doi:10.48550/arXiv.2302.14128.
- 23 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.
- 24 Bart M. P. Jansen, Daniel Lokshantov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.
- 25 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for  $(k, r)$ -center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.
- 26 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discret. Appl. Math.*, 126(2-3):197–221, 2003. doi:10.1016/S0166-218X(02)00198-1.
- 27 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.
- 28 Daniel Lokshantov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.
- 29 Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016. doi:10.1007/978-3-319-40970-2\_1.
- 30 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

- 31 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/s00453-015-0030-x.
- 32 Marcin Pilipczuk. A tight lower bound for vertex planarization on graphs of bounded treewidth. *Discret. Appl. Math.*, 231:211–216, 2017. doi:10.1016/j.dam.2016.05.019.
- 33 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0\_51.



# An Iterative Approach for Counting Reduced Ordered Binary Decision Diagrams

Julien Clément ✉

Normandie Université, UNICAEN, ENSICAEN, CNRS, GREYC – UMR 6072, France

Antoine Genitrini ✉

Sorbonne Université, CNRS, LIP6 – UMR 7606, F-75005 Paris, France

---

## Abstract

For three decades *binary decision diagrams*, a data structure efficiently representing Boolean functions, have been widely used in many distinct contexts like model verification, machine learning, cryptography and also resolution of combinatorial problems. The most famous variant, called reduced ordered binary decision diagram (ROBDD for short), can be viewed as the result of a compaction procedure on the full decision tree. A useful property is that once an order over the Boolean variables is fixed, each Boolean function is represented by exactly one ROBDD. In this paper we aim at computing the *exact distribution of the Boolean functions in  $k$  variables according to the ROBDD size*, where the ROBDD size is equal to the number of decision nodes of the underlying directed acyclic graph (DAG) structure. Recall the number of Boolean functions with  $k$  variables is equal to  $2^{2^k}$ , which is of double exponential growth with respect to the number of variables. The maximal size of a ROBDD with  $k$  variables is  $M_k \approx 2^k/k$ . Apart from the natural combinatorial explosion observed, another difficulty for computing the distribution according to size is to take into account dependencies within the DAG structure of ROBDDs. In this paper, we develop the first polynomial algorithm to derive the distribution of Boolean functions over  $k$  variables with respect to ROBDD size denoted by  $n$ . The algorithm computes the (enumerative) generating function of ROBDDs with  $k$  variables up to size  $n$ . It performs  $O(k n^4)$  arithmetical operations on integers and necessitates storing  $O((k+n)n^2)$  integers with bit length  $O(n \log n)$ . Our new approach relies on a decomposition of ROBDDs layer by layer and on an inclusion-exclusion argument.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Generating functions; Mathematics of computing → Combinatoric problems; Theory of computation → Generating random combinatorial structures; Information systems → Data compression; Theory of computation → Data compression

**Keywords and phrases** Boolean Function, Reduced Ordered Binary Decision Diagram (`{robdd}`), Enumerative Combinatorics, Directed Acyclic Graph

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.36

**Supplementary Material** *Software*: <https://github.com/agenitrini/BDDgen>  
archived at `swh:1:dir:dc717703d5409305685bff27e67735eba792508d`

**Funding** *Julien Clément*: PING/ACK [ANR-18-CE40-0011], C\_SYDiSi [ANR-19-CE48-0007]

**Acknowledgements** The authors thank the anonymous referees for their comments and suggested improvements. All these remarks have increased the quality of the paper.

## 1 Introduction

Three decades ago a central data structure in computer science, designed to represent Boolean functions, emerged under the name of Binary Decision Diagrams (or BDDs) [1]. Their algorithmic paradigm gives great advantages: it is based on a *divide-and-conquer* approach combined with a compaction process. Their benefits compared to other Boolean representations are so obvious that several dozens of BDD variants have been developed



© Julien Clément and Antoine Genitrini;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 36; pp. 36:1–36:15

Leibniz International Proceedings in Informatics



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

in recent years. In his monograph [16], Wegener presents several ones like ROBDDs [2], OKFBDDs [5], QOBDDs [15], ZBDDs [10], and others. While most of these data structures are used in the context of verification [16], they also appear, for example, in the context of cryptography [9] or knowledge compilation [4]. Also, the size of the structure, depending on the compaction of a decision tree, allows improving classification in the context of machine learning [11]. Finally, some specific BDDs are relevant to strategies for the resolution of combinatorial problems, cf. [8, vol. 4], like the classical satisfiability count problem.

The classical way to represent the different diagrams consists in their embedding as directed acyclic graphs (or DAGs). In the following we are interested in the original form of decision diagrams that are ROBDDs, for *Reduced Ordered Binary Decision Diagrams*. One of their fundamental properties relies on the single, thus canonical, representative property for each Boolean function (with a given order over the Boolean variables). In his book [8] Knuth recalls and proves several combinatorial results for ROBDDs. He is, for example, interested in the profile of a typical ROBDD, or in the way to combine two structures to represent a more complex Boolean function. However, thirty years after the takeoff of ROBDDs, the study of the distribution of Boolean functions with respect to the size, defined as the number of decision nodes (see Figure 2), of the DAG structure is not totally understood. The main problem is that no recursive characterization describing the structure of ROBDDs is known, as opposed, for instance, to the recursive decomposition of binary trees which is the core approach in their combinatorial studies (profile, width, depth).

**Related work.** An important step in the comprehension of the distribution of the Boolean functions according to their ROBDD size has been achieved by Wegener [15] and improved by Gröpl *et al.* [7]. These authors proved that almost all functions have the same ROBDD size up to a factor of  $1 + o(1)$  when the number of variables  $k$  tends to infinity, exhibiting the Shannon effect (strong or weak depending on the value of  $k$ ). The strong (respectively weak) Shannon effect states that almost all functions have the same ROBDD size as the largest ROBDDs up to a factor of  $1 + f(k)$  with  $f(k) = o(1)$  (resp.  $f(k) = \Omega(1)$ ) as  $k$  tends to infinity (see also [14]). The reader may find an illustration of this phenomenon in Figure 1 with the plot of the exact distribution for  $k = 13$  variables. A consequence of these first analyses is that picking *uniformly at random* a Boolean function whose ROBDD is small is not an easy task, although in practice ROBDDs are often not of exponential size (with respect to  $k$ ).

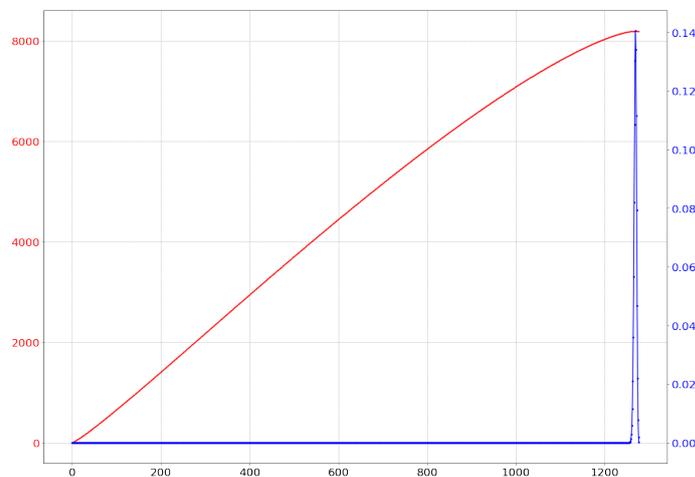
In [12], the authors study, experimentally, numerically, and theoretically, the size of ROBDDs when the number  $k$  of variables is increasing. However, their main approach relies on an exhaustive enumeration of the decision trees of all Boolean functions, that are in a second step compressed into ROBDDs. The doubly exponential growth of Boolean functions over  $k$  variables, equal to  $2^{2^k}$ , allows only to compute the first values for  $k = 1, \dots, 4$ . Then the authors extrapolate the distributions by sampling decision trees (uniformly at random).

Later in the paper [3] we obtain similar combinatorial results. Using a new approach based on a partial recursive decomposition, we partition the ROBDDs according to their profile (which describes the number of nodes per level in the DAG). Another key feature of [3] is that we can restrict ourselves to a maximal size  $n$  for ROBDDs, as opposed to the exhaustive-oriented approach of [12]. Although more efficient, still algorithms with this approach are bound to be at least of complexity  $\Omega(n k^{3/2 \cdot k^2 / \log k})$ , while using a huge amount of extra memory. However, after a lengthy computation we obtain the exact distributions of the size of ROBDDs up to  $k = 9$ , thus partitioning the set of  $2^{512}$  Boolean functions into ROBDDs of sizes ranging from 0 to 141.

**Main results.** In this paper, we describe an algorithm that calculates the exact distribution for ROBDDs up to size  $n$  in time complexity  $O(k n^4)$  using  $O((n+k)n^2)$  extra storage memory for integers. To the best of our knowledge, this is the first polynomial complexity algorithm computing the distribution of the ROBDD size for Boolean functions. Our combinatorial approach is based on an iteration process instead of a recursive approach.

We improve drastically on previous work and all the extrapolated results presented in [12] for Boolean functions up to 13 variables are now fully and exactly described. Using a personal computer, in a couple of minutes we obtain an exhaustive counting of the ROBDDs representing functions over 11 variables. With a computer with several hundreds gigabytes of RAM we compute the distribution over 13 variables in about a day. Indeed, we partition the  $2^{8192}$  Boolean functions according to their ROBDDs size (which ranges from 0 to 1277).

In Figure 1 the exact distribution is depicted in two ways of presentation: a red point  $(x, y)$  states that  $2^y$  functions have a ROBDD size  $x$ , in logarithmic scale; the blue curve is the probability distribution.



■ **Figure 1** The ROBDD size distribution of Boolean functions in 13 variables. The exact distribution is depicted in two ways of presentation: the red curve is the logarithmic scale of the distribution; the blue curve is the probability distribution.

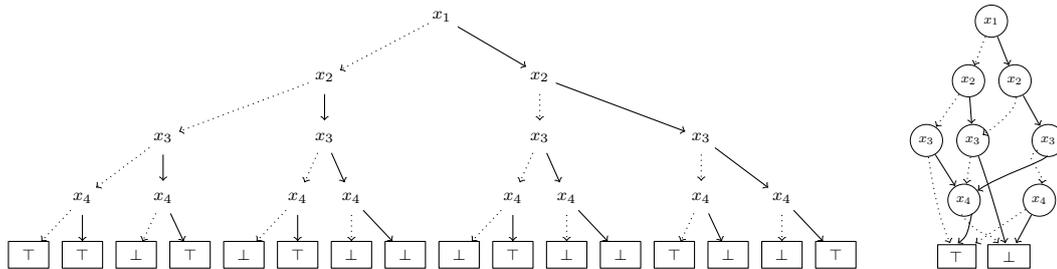
**Organization of the paper.** In Section 2 we present the formal notions and objects necessary for the description of our counting approach. Section 3 presents the iterative process for computing the number of ROBDDs having a given profile (the profile describes the number of decision nodes labelled with each variable). Section 3 also states the main result of this paper which is a formula for computing the number of ROBDDs involving linear maps over a polynomial ring. Finally, Section 4 presents the algorithmic context for computing the complete distribution (under the form of the enumerative generating function [6] of ROBDDs).

## 2 Preliminaries

A Boolean function in  $k$  variables is a function from the set  $\{0, 1\}^k$  into the set  $\{0, 1\}$ . The set of functions is denoted by  $\mathcal{B}_k$  and its cardinality is  $2^{2^k}$ . Furthermore, for the rest of this paper, we choose an ordering of the sequence of variables that corresponds to  $x_1, x_2, \dots, x_k$ . Any other ordering could be chosen, but one must be fixed.

### 2.1 Boolean functions representation

Figure 2 shows a decision tree representing a 4-variable Boolean function and its associated ROBDD. In both structures, traversing a path from the root to a leaf allows to evaluate the function for a given assignment. Being in a node labelled by  $x_i$  and going to its low child (using the dotted edge) corresponds with evaluating  $x_i$  to 0; going to its high child (using the solid edge) corresponds with evaluating  $x_i$  to 1.



■ **Figure 2 (left)** A decision tree and **(right)** its associated ROBDD.

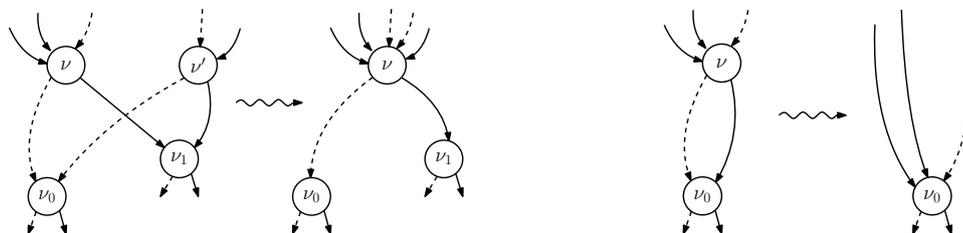
► **Definition 1 (DAG representation).** Let  $f \in \mathcal{B}_k$  be a Boolean function in variables  $x_1, \dots, x_k$ . The function  $f$  can be represented as a rooted directed acyclic graph (DAG for short) composed of internal decision nodes, labelled by variables and two terminal nodes labelled by  $\{\perp, \top\}$  representing respectively the constants 0 and 1. Each decision node labelled by  $x_i$  has two children, the low child (resp. high child) such that traversing the edge to the low child (resp. high child) corresponds to assign  $x_i$  to 0 (resp. to 1). The size of a DAG is its number of decision nodes.

► **Definition 2 (OBDD).** Let  $f \in \mathcal{B}_k$  and pick one of its DAG representation. The DAG is called an Ordered Binary Decision Diagram for  $f$  (OBDD for short) when all paths from the root to a terminal node traverse decision nodes with index in increasing order.

By taking an OBDD for a function with a pointed decision node  $\nu$  labelled by  $x_i$ , and extracting the sub-DAG rooted in  $\nu$  by taking all its descendants, we obtain an OBDD representing a Boolean function in the variables  $x_i, x_{i+1}, \dots, x_k$ .

► **Definition 3 (ROBDD).** Let  $f \in \mathcal{B}_k$  and let  $B$  be one of its OBDD. If all sub-DAGs of  $B$  are representing distinct Boolean functions, then  $B$  is called Reduced Ordered Binary Decision Diagram (ROBDD).

Figure 3 shows the forbidden configurations in ROBDD and the operations used to compress the structure.



■ **Figure 3** The two forbidden configurations in ROBDDs and the resulting operations when compressing: **(left)**  $\nu$  and  $\nu'$  are merged; **(right)**  $\nu$  is deleted (from [7]).

► **Fact.** *Let  $f \in \mathcal{B}_k$ , there exists a single ROBDD representing  $f$ .*

The data structure of ROBDDs is especially famous due to this property of canonicity<sup>1</sup>. The reader may read Knuth [8] for the proof of uniqueness and several other properties satisfied by ROBDDs.

For the rest of the paper we are only dealing with ROBDDs. Furthermore, we take the point of view of layered ROBDD by studying and representing ROBDDs layer by layer: each layer contains all decision nodes labelled by the same variable. This layer-by-layer decomposition is natural since the variables are ordered. Thus, a ROBDD for a function in  $k$  variables is composed of  $k$  layers (plus a layer for the two terminal nodes). Note that a layer could be empty (which means the Boolean function does not depend on the particular variable associated to this layer).

► **Definition 4** (ROBDD's profile). *Let  $f \in \mathcal{B}_k$  and let  $B$  be its ROBDD. Using the layer-by-layer point of view, we define the profile  $B$  to be the non-negative integer sequence  $[p_1, p_2, \dots, p_k]$  such that  $p_i$  is the number of nodes labelled by  $x_i$  in  $B$ , i.e. the size of the layer corresponding to  $x_i$ , for all  $i \in \{1, \dots, k\}$ .*

## 2.2 Combinatorial description

In our previous paper [3] we proposed a kind of recursive decomposition of a ROBDD based on the low and high children of the root. Here we propose a new and simpler point of view, based on a layer-by-layer description, which is much more efficient for the counting problem (also for the generating problem). We need to introduce a generalization of a ROBDD, called multientry ROBDDs, which corresponds exactly to the structure obtained by removing some upper layers in a standard ROBDD (see Figure 4 for an example).

Informally a multientry ROBDD is a structure obtained by cutting off a certain number of the top layers of a ROBDD. The resulting structure is still a DAG, but with several sources. We also keep track of where the half-edges from the top layers were pointing. In Figure 4 a ROBDD and multientry ROBDD obtained by removing the 3 top layers are depicted.

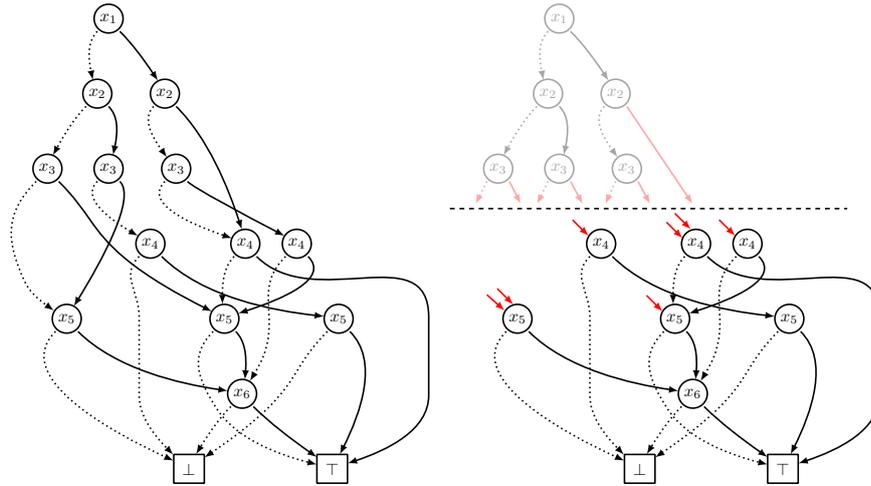
► **Definition 5** (Multientry ROBDD). *A multientry ROBDD  $M$  with at most  $k \geq 0$  variables and size  $n \geq 0$  is a couple  $(M, \mathfrak{E})$ . The structure  $M$  is a layered DAG structure with  $n$  decision nodes  $\mathcal{Q}$  distributed on  $k$  layers and two constant nodes  $\{\top, \perp\}$  such that any two subgraphs are non-identical (as in ROBDDs). The multiset  $\mathfrak{E}$  has its elements in  $\mathcal{Q} \cup \{\top, \perp\}$  and is such that any source node in  $M$  (i.e., having in-degree 0), must appear at least once. The nodes in  $\mathfrak{E}$  are called distinguished (and correspond with destination nodes of red half-edges in Figure 4).*

As a special case, a ROBDD is a multientry ROBDD having one source (the root) and a multiset  $\mathfrak{E}$  reduced to the root (with multiplicity 1).

In the multientry ROBDD in Figure 4, by numbering the nodes from top to bottom and from left to right, i.e. the leftmost  $x_4$  is number 1, the second  $x_4$  is 2, the rightmost  $x_4$  is 3, the rightmost  $x_5$  is 4,  $\dots$ ,  $x_6$  is 7 and  $\perp$  and  $\top$  are respectively 8 and 9, we obtain the multiset  $\mathfrak{E} = \{1, 2, 2, 3, 4, 4, 5\}$ . We note that our definition of multientry ROBDD is similar (but not exactly identical) to the one of *shared*-BDDs presented by Knuth [8] to represent several Boolean functions in the same decision diagram.

Furthermore, remark that the same multientry ROBDD can be exhibited by cutting off top layers (not even the same number) of different ROBDDs.

<sup>1</sup> Uniqueness of the ROBDD for  $f$  is ensured when a variable ordering is fixed.



■ **Figure 4** A ROBDD of size 13 (not counting the constant nodes  $\{\perp$  and  $\top\}$ ), with 6 variables and profile  $\mathbf{p} = [1, 2, 3, 3, 3, 1]$ . **(left)** A DAG representation of the ROBDD; **(right)** Cutting off top three layers, we obtain the multientry ROBDD keeping track with red half edges of nodes which were disconnected.

### 3 Full iterative counting formula

In the following we describe an approach to counting the number of ROBDDs with a given profile  $\mathbf{p}$ . We use a powerful algebraic representation to encapsulate a kind of inclusion-exclusion principle. This motivates the definition of a linear application on polynomials using substitutions. The linearity property of the applications is crucial for achieving our algorithm polynomial complexity. We consider the polynomial ring  $\mathbb{Z}[X]$  and linear endomorphisms over  $\mathbb{Z}[X]$  (i.e., linear maps between  $\mathbb{Z}[X]$  and  $\mathbb{Z}[X]$ ). Thus, for a linear map  $g : \mathbb{Z}[X] \rightarrow \mathbb{Z}[X]$  and two polynomials  $P$  and  $Q$  in  $\mathbb{Z}[X]$ ,  $g[P + Q] = g[P] + g[Q]$ , and for any scalar  $\lambda \in \mathbb{Z}$  and polynomial  $P \in \mathbb{Z}[X]$  we have  $g[\lambda P] = \lambda g[P]$ .

In the following theorem, we state the main result of the paper which gives access to the number of multientry ROBDDs for a given profile.

► **Theorem 6** (Multientry ROBDDs counting formula). *For the family of linear maps  $(\phi_r)_{r \geq 0}$ , each mapping  $\phi_r : \mathbb{Z}[X] \rightarrow \mathbb{Z}[X]$  is defined with respect to the canonical basis  $(X^m)_{m \geq 0}$  by*

$$\phi_r[X^m] = \left( \prod_{i=0}^{r-1} (X^2 - X - i) \right) \cdot \left( \sum_{j=0}^{m-r} \binom{m}{j} \left\{ \begin{matrix} m-j \\ r \end{matrix} \right\} X^j \right). \tag{1}$$

Let  $M(\mathbf{p}, m)$  be the number of multientry ROBDDs with profile  $\mathbf{p} = [p_1, \dots, p_k]$  and  $m$  incoming half edges. We have, for  $k \geq 0$ ,

$$M(\mathbf{p}, m) = (\phi_{\mathbf{p}}[X^m])_{X=2}, \tag{2}$$

where

- $\phi_{\mathbf{p}}$  is the composition product  $\phi_{p_k} \circ \phi_{p_{k-1}} \circ \dots \circ \phi_{p_1}$ ;
- for a polynomial  $P \in \mathbb{Z}[X]$ ,  $(P)_{X=2}$  is the evaluation of  $P$  at  $X = 2$ .

In the theorem,  $\binom{m}{j}$  stands for the binomial coefficient and  $\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$  is the Stirling number of the second kind counting the number ways to partition a set of  $n$  objects into  $k$  non-empty subsets.

► **Remark 7.** This is actually a stronger result than what we need for counting ROBDDs, since the number of ROBDDs corresponds to the special case where  $\mathbf{p} = [p_1 = 1, p_2, \dots, p_k]$  and  $m = 1$  (meaning there is one source node in the top layer). However, the fact that we are able to compute  $\phi_{\mathbf{p}}$  on the basis  $(X^m)_{m \geq 0}$  is key to our approach.

The detailed proof of Theorem 6 is deferred to after an example of such a computation.

► **Example.** Let us consider a profile  $\mathbf{p} = [1, 2, 4, 2]$  for 4 variables  $x_1, x_2, x_3, x_4$  and choosing  $m = 1$  in (2). Then, for  $1 \leq i \leq 4$ , we compute iteratively  $\phi_{p_i} \circ \dots \circ \phi_{p_1}(X)$ :

$$\begin{aligned} X &\stackrel{\phi_1}{\mapsto} X^2 - X \\ &\stackrel{\phi_2}{\mapsto} X^4 - 2X^3 + X \\ &\stackrel{\phi_4}{\mapsto} X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X \\ &\stackrel{\phi_2}{\mapsto} 28X^{10} + 28X^9 - 98X^8 - 112X^7 + 76X^6 + 92X^5 - 12X^4 - 8X^3 + 6X^2. \end{aligned}$$

Evaluating the last polynomial at  $X = 2$ , we get that there are 11 160 ROBDDs with profile  $[1, 2, 4, 2]$ . The power of our approach is that we could have stopped at any iteration, and still get the number of ROBDDs for the considered truncated profile. On the particular example this yields

$\mathbf{p}$	$\phi_{\mathbf{p}}$	$(\phi_{\mathbf{p}})_{X=2}$
$[\ ]$	$X$	2
$[1]$	$X^2 - X$	2
$[1, 2]$	$X^4 - 2X^3 + X$	2
$[1, 2, 4]$	$X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X$	0
$[1, 2, 4, 2]$	$28X^{10} + 28X^9 - 98X^8 - 112X^7 + 76X^6 + 92X^5 - 12X^4 - 8X^3 + 6X^2$	11 160

The number 0 when considering  $[1, 2, 4]$  may seem counterintuitive at first, but indeed a ROBDD can only have up to 2 nodes on its last layer, otherwise one node has to be a duplicate of another.

**Proof of Theorem 6.** (Multientry ROBDDs counting formula). The proof is obtained by induction on the number  $k \geq 0$  of layers with decision nodes.

**Base case.** When  $k = 0$  and  $n \geq 0$ . The number of multientry ROBDDs is  $M([\ ], m) = 2^m$ , i.e.,  $X^m$  evaluated at  $X = 2$ , as we must map the  $m$  half edges to either one of the two constants. Note that if  $m = 0$  then  $M([\ ], m) = 1$  corresponding to the void function (which is a special case).

**Induction step.** Now suppose Theorem 6 is true for  $k \geq 0$ .

Let us consider a profile of length  $k + 1$  as  $[r] \cdot \mathbf{p}$  with  $r \geq 0$  and  $\mathbf{p}$  a profile of length  $k$ .

- If  $r = 0$ , a simple computation shows that  $\phi_0$  is the identity. Hence, the empty layer can in fact be omitted since

$$\phi_{\mathbf{p}} \circ \phi_0 [X^m] = \phi_{\mathbf{p}} [X^m], \quad \text{so that } M([0] \cdot \mathbf{p}, m) = M(\mathbf{p}, m). \tag{3}$$

- From now on let us suppose  $0 < r \leq m$ . The set of  $m$  half edges pointing at the first layer can be decomposed in two subsets for  $j \in \{0, \dots, m - r\}$ :  $j$  entries will go to layers below the first one, and  $m - j$  entries will be mapped to the  $r$  nodes of the first layer. A Stirling number of the second kind  $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$  counts the number of ways to partition a set of  $n$  objects into  $k$  non-empty subsets. So there are  $\binom{m}{j} \cdot \left\{ \begin{smallmatrix} m-j \\ r \end{smallmatrix} \right\}$  such partitions. We write

$$M([r] \cdot \mathbf{p}, m) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ \begin{smallmatrix} m-j \\ r \end{smallmatrix} \right\} f_{\mathbf{p}}^{(r)}(j), \tag{4}$$

## 36:8 An Iterative Approach for Counting ROBDDs

where  $f_{\mathbf{p}}^{(r)}(j)$  denotes the number of multientry ROBDDs with  $j$  free half edges and  $r$  pairs of half edges (the ones resulting from the  $r$  nodes of the first layer). These  $2r$  half edges must thus obey the following constraints:

- in each pair, the two half edges must be distinct, i.e., point to different nodes;
- all  $r$  pairs of half edges must be distinct with one another (as pairs).

Our goal now is to get rid of the constraints coming from these  $r$  nodes and express all quantities in terms of free half edges.

The following equation translates the previous constraints on the pair of adjacent half edges coming from the first of the  $r$  nodes

$$f_{\mathbf{p}}^{(r)}(j) = f_{\mathbf{p}}^{(r-1)}(j+2) - f_{\mathbf{p}}^{(r-1)}(j+1) - (r-1)f_{\mathbf{p}}^{(r-1)}(j). \quad (5)$$

Indeed, the first term  $f_{\mathbf{p}}^{(r-1)}(j+2)$  corresponds in adding 2 free half edges, and results in overcounting. Then following an inclusion-exclusion principle, firstly we subtract  $f_{\mathbf{p}}^{(r-1)}(j+1)$  which would count the number of configurations if the two half edges in this pair were merged. Finally, we subtract  $(r-1)f_{\mathbf{p}}^{(r-1)}(j)$  in (5). This last quantity counts the number of configurations if the pair of half edges was merged with one of the  $r-1$  remaining ones (hence  $r-1$  choices). Note that no additional free half edge is added because of this merge. This yields (5).

Solving the simple recurrence (5) with respect to  $r$  yields

$$f_{\mathbf{p}}^{(r)}(j) = \sum_{i=0}^{2r} a_i f_{\mathbf{p}}^{(0)}(i+j), \quad (6)$$

where coefficients  $(a_i)$  are obtained by identifying  $P(X) = \prod_{i=0}^{r-1}(X^2 - X - i) = \sum_{i=0}^{2r} a_i X^i$ . At this point, we remark the equality true for  $m \geq 0$

$$M(\mathbf{p}, m) = f_{\mathbf{p}}^{(0)}(m), \quad (7)$$

which reflects the fact that in  $M(\mathbf{p}, m)$ , all  $m$  half edges are unconstrained. Then (4) rewrites

$$M([r] \cdot \mathbf{p}, m) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ \begin{matrix} m-j \\ r \end{matrix} \right\} \sum_{i=0}^{2r} a_i M(\mathbf{p}, i+j) = \sum_{i=0}^{m+r} c_i M(\mathbf{p}, i),$$

with coefficients  $c_i$  obtained by identifying  $\sum_{i=0}^{m+r} c_i X^i = \phi_r(X^m)$  from (1).

By induction hypothesis on the length  $k$  of  $\mathbf{p}$  we have for  $0 \leq i \leq m+r$

$$M(\mathbf{p}, i) = (\phi_{\mathbf{p}} [X^i])_{X=2}. \quad (8)$$

By linearity

$$\sum_{i=0}^{m+r} c_i \phi_{\mathbf{p}} [X^i] = \phi_{\mathbf{p}} \left[ \sum_{i=0}^{m+r} c_i X^i \right] = \phi_{\mathbf{p}} [\phi_r [X^m]] = \phi_{\mathbf{p}} \circ \phi_r [X^m],$$

and finally

$$M([r] \cdot \mathbf{p}, m) = \sum_{i=0}^{m+r} c_i M(\mathbf{p}, i) = \sum_{i=0}^{m+r} c_i (\phi_{\mathbf{p}} [X^i])_{X=2} = (\phi_{\mathbf{p}} \circ \phi_r [X^m])_{X=2}$$

This ends the proof. ◀

## 4 Counting algorithms

In this section, we present an algorithm for counting ROBDDs of size  $n$ .

*The time and space complexities are measured respectively in terms of arithmetical operations on  $\mathbb{Z}$  and memory space used to store integers in  $\mathbb{Z}$ . When considering ROBDDs of size upper bounded by  $n$ , all integers in  $\mathbb{Z}$  involved can be checked to be of bit length  $O(n \log n) = O(\log n!)$ .*

The reader can find an implementation of the following algorithms at <https://github.com/agenitrini/BDDgen>.

### 4.1 Linear maps: precomputation step

A first step is to pre-compute a representation of linear maps  $(\phi_r)_{r \geq 0}$ . For a ROBDD of size  $n$  we know that the maximal number of half edges is  $n + 1$ , and the maximal number of nodes on a layer is also loosely upper bound by  $n$ . Hence, it is sufficient to compute  $\phi_r[X^m]$  in  $\mathbb{Z}[X]$  for  $0 \leq r \leq n$  and  $0 \leq m \leq n + 1$ . In the form of (1),  $\phi_r[X^m]$  is equal to  $P_r(X)Q_{r,m}(X)$  with

$$P_r(X) = \prod_{i=0}^{r-1} (X^2 - X - i), \quad \text{and} \quad Q_{r,m}(X) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ \begin{matrix} m-j \\ r \end{matrix} \right\} X^j.$$

So the first step is to compute coefficients of  $P_r(X)$  and  $Q_{r,m}(X)$ . Concerning binomial coefficient and Stirling numbers of the second kind, both tables can be computed by a naive algorithm (for binomials, it is the famous Pascal's triangle) in space  $O(n^2)$  with  $O(n^2)$  arithmetic operations on integers.

Once these coefficients are available, we compute the products  $\phi_r[X^m] = P_r(X)Q_{r,m}(X)$ . Computing  $P_r$  from  $P_{r-1}$  necessitates  $O(n)$  arithmetical operations on integers, yielding a total  $O(n^2)$  number of arithmetical operations for the whole family  $(P_r)_{r \leq n}$ . Each polynomial  $(Q_{r,m})$  necessitates  $O(n)$  arithmetical operations per polynomial (supposing binomial coefficients and Stirling number of the second kind are precomputed). Finally,  $\phi_r[X^m]$  is computed with  $O(n^2)$  arithmetical operations (by a naive product of two polynomials). There are  $O(n^2)$  such polynomials to compute. Hence, we get a total  $O(n^4)$  of arithmetic operations using  $O(n^3)$  storage memory space for coefficients. We thus get the next lemma.

► **Lemma 8** (Precomputing step: linear maps). *The precomputation step for the representation of linear maps by computing  $\phi_r[X^m]$  for  $0 \leq r \leq n$  and  $0 \leq m \leq n + 1$  necessitates  $O(n^4)$  arithmetical operations on integers and uses memory space for  $O(n^3)$  coefficients in  $\mathbb{Z}$ .*

### 4.2 Basic counting

The basic block in our approach is to be able to compute  $\phi_r[P]$  for  $r \geq 0$ . This is done by Algorithm 1 which is a direct translation of Theorem 6 using the linearity of the linear maps  $(\phi_r)_r$ .

► **Proposition 9** (Complexity of basic step). *Let  $P$  be a polynomial of degree  $d$ . Algorithm 1 computes  $\phi_r[P]$  and performs  $O(rd + d^2)$  arithmetical operations over  $\mathbb{Z}$  to compute  $\phi_r[P]$ , using  $O(d^2)$  memory space<sup>2</sup>.*

<sup>2</sup> Recall in Proposition 9 we do not take into account the precomputation step of the family  $(\phi_r[X^m])_r$ .

**Algorithm 1** Computing  $\phi_r[P]$ .

---

**Input:** an integer  $r \geq 0$ , a polynomial  $P(X) = \sum_{m=0}^d p_m X^m \in \mathbb{Z}[X]$   
**Output:**  $\phi_r[P] \in \mathbb{Z}[X]$   
 $Q \leftarrow 0$   
**for**  $m$  **from**  $r$  **to**  $d$  **do**  
    $Q \leftarrow Q + p_m \phi_r[X^m]$   
**return**  $Q$

---

**Algorithm 2** Computing  $(\phi_{\mathbf{p}}[X])_{X=2}$ .

---

**Input:** a profile  $\mathbf{p} = [p_1, \dots, p_k]$   
**Output:** the number of ROBDDs with profile  $\mathbf{p}$   
 $P \leftarrow X$   
**for**  $i$  **from** 1 **to**  $k$  **do**  
    $P \leftarrow \phi_{p_i}[P]$   
**return**  $P(2)$

---

**Proof.** Each polynomial  $\phi_r[X^m]$  is of degree  $r + m = O(r + d)$ . Thus, the number of operations needed on coefficients is  $O(rd + d^2)$  if  $r > 0$  (or  $O(1)$  if  $r = 0$  since  $\phi_0$  is the identity). ◀

By Remark 7 and applying Theorem 6, it is straightforward to compute the number of ROBDDs with profile  $\mathbf{p} = [p_1, \dots, p_k]$  (using  $m = 1$ ). The pseudocode is given in Algorithm 2. We have the following proposition.

► **Proposition 10.** *Algorithm 2 computes the number of ROBDDs of size  $n$  with  $k$  variables and given profile  $\mathbf{p}$ . It performs  $O(k n^2)$  arithmetical operations over  $\mathbb{Z}$  and uses  $O(n)$  extra memory space to store integers.*

**Proof.** The number of ROBDDs is  $(\phi_{\mathbf{p}}[X])_{X=2}$ . The polynomial  $\phi_{\mathbf{p}}[X]$  is computed by iterating  $k$  times a linear map of type  $\phi_r$  starting from an initial polynomial  $X$ . By Proposition 9, the algorithm performs  $O(n^2)$  operations for each iteration since polynomials have  $O(n)$  coefficients. The total computation thus performs  $O(k n^2)$  arithmetical operations over  $\mathbb{Z}$  and use  $O(n)$  memory space to store coefficients. Evaluating  $(\phi_{\mathbf{p}}[X^m])_{X=2}$  at  $X = 2$  can be done in time complexity  $O(n)$  (by Horner's method for instance). ◀

### 4.3 Generating function for ROBDD size

The main goal of this section is compute the distribution of Boolean functions in  $k$  variables according to the ROBDD size. For  $f \in \mathcal{B}_k$  a Boolean function, we let  $\lambda(f)$  be the size of its ROBDD, i.e., its number of decision nodes. A convenient way to represent the distribution of the size  $\lambda$  on  $\mathcal{B}_k$  consists in computing the generating function [6]

$$F_k(u) = \sum_{f \in \mathcal{B}_k} u^{\lambda(f)} = \sum_{i \geq 0} f_i u^i,$$

where  $u$  is a formal variable marking the size. Then for  $i \geq 0$ , the coefficient  $f_i = [u^i]F_k(u)$  is the number of ROBDDs of size  $i$  with  $k$  variables, i.e. the notation  $[u^i]F_k(u)$  corresponds to the coefficient-extraction of the monomial  $u^i$ . We also introduce the truncation  $F_k^{\leq n}(u) = \sum_{0 \leq i \leq n} f_i u^i$  as the generating functions of ROBDDs of size less than or equal to  $n$ .

We first extend the formalism introduced in Section 3 and define a linear map  $\varphi : \mathbb{Z}[u, X] \rightarrow \mathbb{Z}[u, X]$ .

► **Definition 11.** *The linear map  $\varphi : \mathbb{Z}[u, X] \rightarrow \mathbb{Z}[u, X]$  is defined via its action on the basis  $(u^r X^m)_{r, m \geq 0}$  as*

$$\varphi : u^r X^m \mapsto \varphi[u^r X^m] = \sum_{i=0}^{m+1} u^{r+i} \phi_i[X^m]. \quad (9)$$

With this notation we have the following proposition.

► **Proposition 12** (Generating function for ROBDD size). *The generating function enumerating Boolean functions by considering the ROBDD size is given by*

$$F_k(u) = (\varphi^k[X])_{X=2}, \quad \text{where } \varphi^k \text{ denotes the composition product } \underbrace{\varphi \circ \dots \circ \varphi}_{k \text{ times}}$$

**Proof.** Each application of  $\varphi$  corresponds with adding a layer. The formal variable  $u$  marks the number of decision nodes added on the current layer. ◀

We remark that in practice we can truncate polynomials, keeping only the terms useful along the computation. The key point here is that if we consider ROBDDs with size bounded by  $n$ , we should make all computation modulo  $u^{n+1}$ . Indeed, the formal variable  $u$  marks the number of nodes (which is bounded by  $n$ ). Sections A.1 and A.2 in the appendix illustrate this point.

Algorithm 3 computes the bivariate polynomial  $\varphi[X^m]$  for  $m \geq 0$ . Algorithm 4, computes recursively the iterated (univariate) version  $(\varphi^\ell[X^m])_{X=2}$  (that is the evaluation at  $X = 2$ ).

■ **Algorithm 3** Computing  $\varphi[X^m]$ .

---

**Input:** An integer  $m \geq 0$   
**Output:** Returns  $\varphi[X^m] \in \mathbb{Z}[u, X]$   
 $Q \leftarrow 0$  ▷  $Q \in \mathbb{Z}[u]$   
**for**  $r$  **from** 0 **to**  $m$  **do**  
     $Q \leftarrow Q + u^r \phi_r[X^m]$   
**return**  $Q$

---

■ **Algorithm 4** Computing  $(\varphi^\ell[X^m])_{X=2}$ .

---

**Input:** Two integers  $\ell, m$   
**Output:** Returns  $(\varphi^\ell[X^m])_{X=2} \in \mathbb{Z}[u]$   
 ▷ N.B.: Computations done modulo  $u^{n+1}$   
 where  $n$  is the maximal size for ROBDDs  
**if**  $\ell = 0$  **then return**  $2^m$  ▷ base case  
 $Q \leftarrow 0$  ▷  $Q \in \mathbb{Z}[u]$   
 $R \leftarrow \varphi(X^m)$  ▷ Call Alg. 3  
**for**  $j$  **from** 0 **to**  $\deg_X(R)$  **do**  
     $M \leftarrow (\varphi^{\ell-1}[X^j])_{X=2}$  ▷ Call Alg. 4  
     $N \leftarrow [X^j] R(u, X)$  ▷  $N \in \mathbb{Z}[u]$   
     $Q \leftarrow Q + M \cdot N$   
**return**  $Q$

---

► **Lemma 13.** *Algorithm 3 computes  $\varphi[X^m] \in \mathbb{Z}[X, u]$ , which has  $O(n^2)$  integer coefficients. It performs  $O(n^2)$  arithmetical operations over  $\mathbb{Z}$ .*

*Algorithm 4 computes  $(\varphi^\ell[X^m])_{X=2} \in \mathbb{Z}[u]$ , which has  $O(n)$  coefficients. If we omit recursive calls, it performs  $O(n^3)$  arithmetical operations over  $\mathbb{Z}$ , using memoization techniques with  $O((n+k)n^2)$  extra memory storage.*

**Proof.** For Algorithm 3, we perform  $m = O(n)$  additions of polynomials in  $\mathbb{Z}[X, u]$  with  $O(n)$  terms, yielding  $O(n^2)$  arithmetical operations over  $\mathbb{Z}$ . The result is a bivariate polynomial of bounded degree ( $n$  for the variable  $u$ ,  $2n$  for the variable  $X$ ) yielding  $O(n^2)$  coefficients. We suppose Algorithm 3 has access to polynomials  $\phi_r[X^m] \in \mathbb{Z}[X]$  from a precomputation step.

For Algorithm 4, two ingredients are essential. Firstly we have to truncate polynomials modulo  $u^{n+1}$  so that operations on univariate polynomials have complexity  $O(n)$  for addition and  $O(n^2)$  for multiplication (using the naive multiplication on polynomials). Secondly we also use memoization techniques (meaning we compute in lazy manner intermediate results only once and keep it for further reference, at the expense of memory storage). That means that we consider that at the time we compute  $(\varphi^\ell[X^m])_{X=2}$ , the polynomials  $(\varphi^{\ell-1}[X^j])_{X=2}$  are available (i.e., their complexity is taken into account independently). By an amortizing argument, the complexity of computing the complete family of polynomials  $(\varphi^\ell[X^m])_{X=2}$

## 36:12 An Iterative Approach for Counting ROBDDs

$(0 \leq \ell \leq k$  and  $0 \leq m \leq n + 1)$  is still  $O(n^3)$  arithmetical coefficients per polynomial. We need to store  $O(kn^2)$  integer coefficients for memoization of all intermediate polynomials. We also suppose Algorithm 4 has access to bivariate polynomials  $(\varphi[X^m] \in \mathbb{Z}[u, X])_{0 \leq m \leq n+1}$  from a precomputation step which requires  $O(n^3)$  memory space for integer coefficients. A subtle point is to understand that we can truncate polynomials at each step in Algorithm 4 and still get the correct result: this can be proved by recurrence on  $\ell$  (see also Section A.2 of the appendix for an example). ◀

Algorithm 4 also computes the generating function of ROBDDs for size up to  $n$  since posing  $\ell = k$  and  $m = 1$  we have

$$F_k^{\leq n}(u) = (\varphi^\ell[X])_{X=2} \pmod{u^{n+1}}.$$

► **Theorem 14** (Algorithm for computing the exact distribution). *We compute the generating function  $F_k^{\leq n}(u)$  of Boolean functions in  $\mathcal{B}_k$  for ROBDDs of size less than or equal to  $n$  using  $O(k n^4)$  arithmetical operations in  $\mathbb{Z}$  and  $O((k+n)n^2)$  for memory space storing integers.*

**Proof.** From Lemmas 13 and 8, the overall complexity is dominated by the computation of the family  $(\phi_r[X^m])_{r,m}$  and the calls to Algorithm 4. By Lemma 13, each polynomial  $(\varphi^\ell[X^j])_{X=2}$  is computed with  $O(n^3)$  arithmetic operations on integer coefficients and there are  $O(k n)$  such polynomials. Hence, the total number operations over  $\mathbb{Z}$  is  $O(k n^4)$ . Furthermore, we store  $O(k n^2)$  coefficients in  $\mathbb{Z}$  for memoization. We also store  $O(n^3)$  coefficients for the family  $(\phi_r[X^m])_{m,r} \in \mathbb{Z}[X]$ . This yields the claimed complexity. ◀

To evaluate the complete size distribution we need to consider the size of largest ROBDDs with  $k$  variables.

► **Theorem 15** (Maximal size of ROBDDs). *Let  $k \geq 1$  be an integer, the maximal number of nodes in a ROBDD with at most  $k$  variables is*

$$M_k = 2^{k-\theta} - 3 + 2^{2^\theta}, \quad \text{with } \theta = \lfloor \log_2(k - \lfloor \log_2(k) \rfloor) \rfloor.$$

*The generating function  $F_k(u)$  of Boolean functions in  $\mathcal{B}_k$  according to the ROBDD size can be computed with  $O(2^{4k}/k^3)$  arithmetical operations in  $\mathbb{Z}$  and uses  $O(2^{2k}/k)$  space.*

**Proof.** Note that this formula is equivalent to the one given without proof by Pontus von Brömssen [13]. The existence of  $\theta$  is proved in [12] and from there we can derive the explicit expression of  $\theta$  (details omitted here). Then substituting  $n = M_k$  in Theorem 14, and noting that  $M_k \approx 2^k/k$  yields the computational complexity result. ◀

Note this is the polynomial with respect to the maximal size of a ROBDD for  $k$  variables, hence a huge improvement compared to exponential brute force algorithms enumerating all  $2^{2^k}$  Boolean functions and computing their ROBDD size obtained after a compaction process. With a careful implementation, we can achieve the computation of  $F_k(u)$  for  $k$  up to 11 variables on a personal computer, and, on a high-performance computer with 512 GB RAM memory, for  $k = 12$  in less than 1 hour 30 minutes, and even for  $k = 13$  in less than 30 hours using the PyPy implementation of Python.

## 5 Conclusion

As an application of the counting approach of this paper we are able sample at random ROBDDs. More precisely, we can efficiently and uniformly pick ROBDDs either according to a given size, or even a given profile or a given spine. This is a great improvement when comparing to the classical uniform random generation over the set of Boolean functions, like in [12], that is drastically biased to the largest ROBDDs due to the Shannon effect. For instance with 12 variables, the probability of drawing uniformly a Boolean function giving a ROBDD of (quadratic in  $k$ ) size  $144 = 12^2$  is approximately  $1.212 \cdot 10^{-957}$ .

In practice, several classical functions have ROBDDs of small size. For example the symmetrical functions in  $k$  variables are associated with ROBDDs of quadratic size in  $k$  (see [8]). Hence, the approach described in this paper leads the way to provide (polynomial) uniform random generator for ROBDDs of small size (i.e., of size less than exponential).

An interesting future work consists in enumerating the BDD structures where our counting and sampling methods can be applied such as (non-reduced) OBDD, or ZDD which generally used to represent sets.

Finally, another research direction consists in noting that the generating function of ROBDDs with both size and number of variables can be specified thanks to an iterative process as in Theorem 12. It would be interesting to see if the machinery of analytic combinatorics [6] is amenable to this kind of specification.

---

## References

- 1 R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
- 2 R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
- 3 J. Clément and A. Genitrini. Binary decision diagrams: From tree compaction to sampling. In *LATIN: Theoretical Informatics – 14th Latin American Symposium, Proceedings*, volume 12118 of *LNCS*, pages 571–583. Springer, 2020.
- 4 A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Int. Res.*, 17(1):229–264, September 2002.
- 5 R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams. In *DAC'94*, pages 415–419, 1994.
- 6 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.
- 7 C. Gröpl, H. J. Prömel, and A. Srivastav. Ordered binary decision diagrams and the shannon effect. *Discret. Appl. Math.*, 142(1-3):67–85, 2004. doi:10.1016/j.dam.2003.02.003.
- 8 D. E. Knuth. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms*. Addison-Wesley Professional, 2011.
- 9 L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure Function Evaluation with Ordered Binary Decision Diagrams. In *CCS'06*, pages 410–420. ACM, 2006.
- 10 S.-I. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. *30th ACM/IEEE Design Automation Conference*, pages 272–277, 1993.
- 11 C. Mues, B. Baesens, C. M. Files, and J. Vanthienen. Decision diagrams in machine learning: an empirical study on real-life credit-risk data. *Expert Systems with Applications*, 27(2):257–264, 2004.
- 12 J. Newton and D. Verna. A theoretical and numerical analysis of the worst-case size of reduced ordered binary decision diagrams. *ACM TCL*, 20(1):6:1–6:36, 2019.

- 13 N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, September 2019. Sequence A327461.
- 14 J. Vuillemin and F. Béal. On the BDD of a Random Boolean Function. In *ASIAN'04*, pages 483–493, 2004.
- 15 I. Wegener. The size of reduced OBDDs and optimal read-once branching programs for almost all Boolean functions. In *GTCCS'94*, pages 252–263, 1994.
- 16 I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

## A Appendix

### A.1 Iterating $\varphi$ (an example)

In this section, we illustrate how  $\varphi$  can be iterated to count the distribution of ROBDDs up to 4 variables. Let us consider the set of Boolean function  $\mathcal{B}_k$  for  $k \in \{1, 2, 3\}$  and pose  $H_k(u, X) = \varphi^k[X]$ .

- $H_1(u, X) = \varphi[X] = (\phi_0(X) + u\phi_1(X)) = (X^2 - X)u + X^2$ . We can verify that substituting  $X = 2$  we get  $F_1(u) = H_1(u, 2) = 2u + 2$ . Indeed,  $\mathcal{B}_1 = \{\perp, x_1, \overline{x_1}, \top\}$ , with respective truth tables  $\{00, 01, 10, 11\}$  leading to 2 ROBDDs of size 1 and 2 ROBDDs of size 0, i.e., with no decision node.
- Adding a second layer, we get  $H_2(u, X) = \varphi(H_1(u, X)) = \varphi^2[X]$ , yielding

$$H_2(u, X) = (X^4 - 2X^3 + X)u^3 + 2(X^3 - X^2)u^2 + 2(X^2 - X)u + X.$$

We get  $F_2(u) = 2u^3 + 8u^2 + 4u + 2$ , hence there are 2, 8, 4, 2 ROBDDs of respective sizes 3, 2, 1 and 0 for 16 Boolean functions on 2 variables.

- Iterating with a third layer, we get  $H_3(u, X) = \varphi(H_2(u, X)) = \varphi^3[X]$  which has 34 terms and gives

$$F_3(u) = 74u^5 + 88u^4 + 62u^3 + 24u^2 + 6u + 2.$$

Hence, there are respectively 74, 88, 62, 24, 6 and 2 ROBDDs of size 5, 4, 3, 2, 1 and 0.

- Adding a fourth layer yields for  $H_4(u, X) = \varphi^4[X]$  a polynomial with 134 terms and

$$F_4(u) = 11160u^9 + 23280u^8 + 17666u^7 + 8928u^6 + 3248u^5 + 960u^4 + 236u^3 + 48u^2 + 8u + 2.$$

There are 11160 ROBDDs with 4 variables of size 9, 23280 ROBDDs of size 8, etc.

Of course, we can check that  $F_k(1) = 2^{2^k}$ , which is the number of Boolean functions on  $k$  variables.

### A.2 Truncating polynomials (an example)

In this subsection, we illustrate on an example the computational effect of truncating polynomials.

Let us consider that the generating function  $F_3^{\leq 3}(u)$  of ROBDD with  $k = 3$  variables and with less or equal to  $n = 3$  decision nodes. The maximal size of a ROBDD for  $k = 3$  variables is  $M_3 = 7$  (7 decision nodes). Then  $\varphi^3[X]$  is a polynomial with 34 terms. We can write (terms who would disappear modulo  $u^{n+1} = u^4$  are grayed)

$$\begin{aligned}
\varphi^3[X] &= (X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X)u^7 \\
&\quad + 4(X^7 - 2X^6 - 3X^5 + 5X^4 + 4X^3 - 3X^2 - 2X)u^6 \\
&\quad + (8X^6 - 12X^5 - 11X^4 + 14X^3 + 4X^2 - 3X)u^5 \\
&\quad + 2(5X^5 - 6X^4 - 5X^3 + 4X^2 + 2X)u^4 \\
&\quad + (9X^4 - 10X^3 - 2X^2 + 3X)u^3 \\
&\quad + 6(X^3 - X^2)u^2 \\
&\quad + 3(X^2 - X)u \\
&\quad + X
\end{aligned}$$

Truncating modulo  $u^4$  and substituting  $X = 2$  yields the polynomial

$$F_3^{\leq 4}(u) = 62u^3 + 24u^2 + 6u + 2.$$

The problem of computing all the terms before truncating the result is that there are  $O(\frac{2^{2k}}{k^2})$  terms (since  $M_k \approx 2^k/k$ ), hence a combinatorial explosion.

In contrast, Algorithm 4 works in a recursive manner and truncate polynomials along the way so that we have a polynomial number of terms. In the following, we describe with some details how to compute  $(\varphi^3(X))_{X=2}$ .

First, Algorithm 4 decomposes  $(\varphi^3(X))_{X=2}$  as  $(\varphi^2 \circ \varphi[X])_{X=2}$ . In general,  $\varphi[X^m]$  is a polynomial of respective degree  $m$  and  $2m$  in variables  $u$  and  $X$  and has  $O(m^2)$  integer coefficients. We compute (collecting terms with respect to variable  $X$ )

$$\varphi[X] = (X^2 - X)u + X = X^2u + X(1 - u). \quad (10)$$

Denoting  $B_m^{(2)}(u) = (\varphi^2[X^m])_{X=2}$ , our algorithm computes recursively for the basis  $(1, X, X^2)$  (removed monomials modulo  $u^4$  are grayed)

$$\begin{aligned}
B_0^{(2)}(u) &= 1 \\
B_1^{(2)}(u) &= 2u^3 + 8u^2 + 4u + 2 \\
B_2^{(2)}(u) &= 74u^4 + 90u^3 + 68u^2 + 20u + 4
\end{aligned}$$

Then since  $\varphi$  is linear, we compute (still modulo  $u^4$ ) using Equation (10)

$$\begin{aligned}
(\varphi^3(X))_{X=2} &= B_2(u)u + B_1(u)(1 - u) \\
&= u(90u^3 + 68u^2 + 20u + 4) + (1 - u)(2u^3 + 8u^2 + 4u + 2) \\
&= 88u^4 + 62u^3 + 24u^2 + 6u + 2.
\end{aligned}$$

In summary, computing modulo  $u^{n+1}$  along the process allows us to control the degree  $O(n)$  of polynomials involved, which in turn ensures that the computation stays of polynomial complexity (with respect to arithmetical operations on integers).

Observing such curves for  $k$  from 1 to 13, we notice the exponential growth of the largest ROBDDs when the number  $k$  of variables increases. Indeed, in Theorem 15 we define  $M_k$  to be the size of the largest ROBDDs with  $k$  variables. The sequence starts as  $(M_k)_{k=1, \dots, 13} = (1, 3, 5, 9, 17, 29, 45, 77, 141, 269, 509, 765, 1277)$ .



# Inductive Continuity via Brouwer Trees

Liron Cohen   

Ben-Gurion University, Beer-Sheva, Israel

Bruno da Rocha Paiva   

University of Birmingham, UK

Vincent Rahli   

University of Birmingham, UK

Ayberk Tosun   

University of Birmingham, UK

---

## Abstract

Continuity is a key principle of intuitionistic logic that is generally accepted by constructivists but is inconsistent with classical logic. Most commonly, continuity states that a function from the Baire space to numbers, only needs approximations of the points in the Baire space to compute. More recently, another formulation of the continuity principle was put forward. It states that for any function  $F$  from the Baire space to numbers, there exists a (dialogue) tree that contains the values of  $F$  at its leaves and such that the modulus of  $F$  at each point of the Baire space is given by the length of the corresponding branch in the tree. In this paper we provide the first internalization of this “inductive” continuity principle within a computational setting. Concretely, we present a class of intuitionistic theories that validate this formulation of continuity thanks to computations that construct such dialogue trees internally to the theories using effectful computations. We further demonstrate that this inductive continuity principle implies other forms of continuity principles.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Type theory; Theory of computation  $\rightarrow$  Constructive mathematics

**Keywords and phrases** Continuity, Dialogue trees, Stateful computations, Intuitionistic Logic, Extensional Type Theory, Constructive Type Theory, Realizability, Theorem proving, Agda

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.37

**Supplementary Material** *Software*: <https://github.com/vrahli/opentt>  
archived at `swh:1:dir:a40178de24063c6bef43e49eb5478ec063a93c90`

**Funding** *Liron Cohen*: This research was partially supported by Grant No. 2020145 from the United States-Israel Binational Science Foundation (BSF).

**Acknowledgements** We would like to thank Martin Escardo, Martin Baillon, and Yannick Forster for useful discussions about continuity and dialogue trees.

## 1 Introduction

The continuity principle is a cornerstone in intuitionistic theories which is generally accepted by constructivists but contradicts classical mathematics. In essence, the principle states that functions on the Baire space (i.e.,  $\mathfrak{B} \equiv \text{Nat} \rightarrow \text{Nat}$ ) only need finite inputs, i.e., initial segments of points of the Baire space, to produce outputs. Different variants of the continuity principle have been developed to capture different levels of strictness in the notion of continuity and different computational aspects. Perhaps the most common continuity principle is the continuity principle for numbers, sometimes referred to as the weak continuity principle (WCP) [24, 15, 4, 7, 36]. WCP states that given a function  $F \in \mathfrak{B} \rightarrow \text{Nat}$  and an point  $\alpha$  of the Baire space  $\mathfrak{B}$ ,  $F(\alpha)$  can only depend on an initial segment of  $\alpha$ , and the



© Liron Cohen, Bruno da Rocha Paiva, Vincent Rahli, and Ayberk Tosun;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 37; pp. 37:1–37:16

Leibniz International Proceedings in Informatics



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

## 37:2 Inductive Continuity via Brouwer Trees

length of the smallest such segment is the modulus of continuity of  $F$  at  $\alpha$ . This is standardly formalized as follows, where  $\mathfrak{B}_n \equiv \{x : \text{Nat} \mid x < n\} \rightarrow \text{Nat}$  is the set of finite sequences of length  $n$ :

$$\text{WCP} \equiv \prod F : \mathfrak{B} \rightarrow \text{Nat} . \prod \alpha : \mathfrak{B} . \|\Sigma n : \text{Nat} . \prod \beta : \mathfrak{B} . (\alpha = \beta \in \mathfrak{B}_n) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})\|$$

However, as shown, e.g., by Kreisel [25, p.154], Troelstra [38, Thm.IIA], and Escardó and Xu [18, 40], continuity is not an extensional property in the sense that two (extensionally) equal functions might have different moduli of continuity. Therefore, to computationally realize continuity, the existence of a modulus of continuity has to be truncated as explained, e.g., in [18, 40, 32, 33], which is what the  $\|\_ \|\_$  operator achieves in  $\text{WCP}$ 's definition above.

Brouwer used  $\text{WCP}$ , along with a consequence of Bar Induction called the Fan Theorem, to derive the following uniform continuity principle ( $\text{UCP}$ ) [7, p.113], which he then used to prove that all real-valued function on the unit interval are uniformly continuous [24, 15, 4, 7, 36], where  $\mathfrak{C} \equiv \text{Nat} \rightarrow \text{Bool}$  is the Cantor space and  $\mathfrak{C}_n \equiv \{x : \text{Nat} \mid x < n\} \rightarrow \text{Bool}$ :

$$\text{UCP} \equiv \prod F : \mathfrak{C} \rightarrow \text{Nat} . \Sigma n : \text{Nat} . \prod \alpha, \beta : \mathfrak{C} . (\alpha = \beta \in \mathfrak{C}_n) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})$$

Note that  $\text{UCP}$  does not need to be truncated as shown for example in [18].

Another version of the continuity principle, which originates from the completeness of Brouwer's bar thesis and implies both  $\text{WCP}$  and  $\text{UCP}$ , has been recently studied [20, 22, 21, 19]. This principle, referred to here as the Inductive Continuity Principle ( $\text{ICP}$ ), relies on a notion of dialogue trees related to Brouwer trees [36] and reminiscent of Kleene trees [23]. This tree-based technique of capturing continuity information, pioneered in [20, 22, 21, 19], and reused for example in [9, 35, 3], consists in computing a tree that, given a function  $F$  from a subset of  $\mathfrak{B}$  to numbers, contains the values of  $F$  at its leaves, and such that the amount of information needed to compute these values, i.e., the modulus of continuity of  $F$  at each point, is given by its branches. This can be formalized as follows where  $\mathfrak{B}_{\text{SNat}} \equiv \text{Nat} \rightarrow \text{SNat}$  for  $\text{SNat}$  a subtype of  $\text{Nat}$  ( $\text{Bt}$  and  $\text{follow}(d, \alpha)$  are made formal in Sec. 3.2).<sup>1</sup>

$$\text{ICP} \equiv \prod F : \mathfrak{B}_{\text{SNat}} \rightarrow \text{Nat} . \|\Sigma d : \text{Bt} . \prod \alpha : \mathfrak{B}_{\text{SNat}} . \text{follow}(d, \alpha) = F(\alpha) \in \text{Nat}\|$$

A number of theories have been shown to satisfy Brouwer's continuity principle, or uniform variants, such as  $\text{N-HA}^\omega$  by Troelstra [37, p.158],  $\text{MLTT}$  by Coquand and Jaber [12, 13],  $\text{System T}$  by Escardó [19],  $\text{MLTT}$  by Xu [40],  $\text{CTT}$  by Rahli and Bickford [32],  $\text{BTT}$  by Baillon, Mahboubi and Pedrot [3], among others (see Sec. 6 for details). These proofs often rely on a semantic forcing-based approach [12, 13], where the forcing conditions capture the amount of information needed when applying a function to a sequence in the Baire space, or through suitable models that internalize ( $\text{C-Spaces}$  in [41]) or exhibit continuous behavior (e.g., dialogue trees in [19, 3]).

Not only can functions on the Baire space be proved to be continuous, but using effectful computations one can in fact *compute* their modulus of continuity [28]. The  $\text{TT}_{\mathfrak{C}}^\square$  family of effectful extensional type theories, recalled in Sec. 2, was shown to be consistent with a version of  $\text{WCP}$  using a family of realizability models that allow validating this principle using effectful computations, and in particular using reference cells [11]. Building on this result, in this paper we identify a family of effectful type theories that are consistent with a variant of  $\text{ICP}$ , and prove this consistency result using effectful computations, namely references.

<sup>1</sup> We use here Brouwer trees, which are equivalent to dialogue trees for functions on the Baire space [17].

$v \in \text{Value}$	::=	$vt$ (type)		$\lambda x.t$ (lambda)		$\star$ (constant)
		$\underline{n}$ (number)		$\text{inl}(t)$ (left injection)		$\boxed{\delta}$ (choice name)
		$\langle t_1, t_2 \rangle$ (pair)		$\text{inr}(t)$ (right injection)		
$vt \in \text{Type}$	::=	$\prod x:t_1.t_2$ (product)		$\{x : t_1 \mid t_2\}$ (set)		$t_1+t_2$ (disjoint union)
		$\sum x:t_1.t_2$ (sum)		$t_1=t_2 \in t$ (equality)		$\ t\ $ (truncation)
		$\mathbb{U}_i$ (universe)		$\text{Nat}$ (numbers)		$\boxed{\text{pure}}$ (pure)
		$t_1 \cap t_2$ (intersection)				
$t \in \text{Term}$	::=	$x$ (variable)		$\boxed{!t}$ (read)		$t_1 <? t_2$ (less than)
		$v$ (value)		$\boxed{\nu x.t}$ (fresh)		$t_1 =? t_2$ (equality)
		$t_1 t_2$ (application)		$\boxed{t_1 := t_2}$ (write)		$\text{let } x = t_1 \text{ in } t_2$ (call-by-value)
		$\text{fix}(t)$ (fixpoint)		$t_1 + t_2$ (addition)		$\text{let } x, y = t_1 \text{ in } t_2$ (pair destructor)
		$\text{case } t \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2$ (injection destructor)				

---

$(\lambda x.t) u$	$w \mapsto_w t[x \setminus u]$	$\underline{n} <? \underline{m} \quad w \mapsto_w \text{inl}(\star), \text{ if } n < m$
$\text{fix}(v)$	$w \mapsto_w v \text{ fix}(v)$	$\underline{n} <? \underline{m} \quad w \mapsto_w \text{inr}(\star), \text{ if } n \not< m$
$\text{let } x = v \text{ in } t_2$	$w \mapsto_w t_2[x \setminus v]$	$\underline{n} =? \underline{m} \quad w \mapsto_w \text{inl}(\star), \text{ if } n = m$
$\text{let } x, y = \langle t_1, t_2 \rangle \text{ in } t$	$w \mapsto_w t[x \setminus t_1; y \setminus t_2]$	$\underline{n} =? \underline{m} \quad w \mapsto_w \text{inr}(\star), \text{ if } n \neq m$
		$\underline{n} + \underline{m} \quad w \mapsto_w n + m$

$! \delta$	$w \mapsto_w \text{read}(w, \delta)$	
$\delta := t$	$w \mapsto_{\text{write}(w, \delta, t)} \star$	$\text{case } \text{inl}(t) \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2 \quad w \mapsto_w t_1[x \setminus t]$
$\nu x.t$	$w \mapsto_{\text{start} \nu \mathcal{C}(w)} t[x \setminus \nu \mathcal{C}(w)]$	$\text{case } \text{inr}(t) \text{ of } \text{inl}(x) \Rightarrow t_1 \mid \text{inr}(y) \Rightarrow t_2 \quad w \mapsto_w t_2[y \setminus t]$

■ **Figure 1** Core syntax (above) and small-step operational semantics (below).

Importantly, in addition to validating the continuity of  $\text{TT}_{\mathcal{C}}^{\square}$  functions using dialogue trees, our work provides the first internalization of the principle into a computational system in the sense that we extend  $\text{TT}_{\mathcal{C}}^{\square}$  with a variant of ICP in Sec. 3, and exhibit in Sec. 5 an effectful  $\text{TT}_{\mathcal{C}}^{\square}$  program that realizes this axiom. The most challenging aspect of internalizing this dialogue-based technique is in proving termination of the computation of such trees. We further show in Sec. 4 that ICP encompasses both weak and uniform continuity. It is however still unknown whether ICP is in fact strictly stronger than the other principles.

## 2 Background

This section reviews  $\text{TT}_{\mathcal{C}}^{\square}$  [10] – a family of extensional type theories parameterized by a choice operator  $\mathcal{C}$  and a metatheoretical modality  $\square$ , which allows typing the choice operators.

### 2.1 Metatheory

Our metatheory is Agda’s type theory [2]. The results presented in this paper have been formalized in Agda: <https://github.com/vrahli/opentt/>. We use  $\forall, \exists, \wedge, \vee, \rightarrow, \neg$  in place of Agda’s logical connectives in this paper, and use  $\top$  for True and  $\perp$  for False. Agda provides a hierarchy of types annotated with universe labels which we omit for simplicity. Following Agda’s terminology, we refer to an Agda type as a *set*, and reserve the term *type* for  $\text{TT}_{\mathcal{C}}^{\square}$ ’s types. We use  $\mathbb{P}$  as the type of sets that denote propositions;  $\mathbb{N}$  for the set of natural numbers; and  $\mathbb{B}$  for the set of Booleans true and false. We use induction-recursion to define the forcing interpretation in Sec. 2.3, where we use function extensionality to interpret universes. We also use classical reasoning twice in the proof presented in Sec. 5.

### 2.2 $\text{TT}_{\mathcal{C}}^{\square}$ ’s Syntax and Operational Semantics

Fig. 1 recalls  $\text{TT}_{\mathcal{C}}^{\square}$ ’s syntax and operational semantics, where the blue boxes highlight the effectful components, and where  $x$  belongs to a set of variables  $\text{Var}$ . For simplicity, numbers are considered to be primitive and the constant  $\star$  is used in place of a term when the particular term used is irrelevant. We use all letters as metavariables for terms and denote

by  $t[x \setminus u]$  the capture-avoiding substitution of all the free occurrences of  $x$  in  $t$  by  $u$ . We write **if**  $t_1$  **then**  $t_2$  **else**  $t_3$  for **case**  $t_1$  of **inl**( $x$ )  $\Rightarrow t_2$  | **inr**( $x$ )  $\Rightarrow t_3$ , where  $x$  does not occur in  $t_2$  or  $t_3$ , and  $t_1; t_2$  for **let**  $x = t_1$  **in**  $t_2$  where  $x$  does not occur free in  $t_2$ .

Types are syntactic forms that are given semantics in Sec. 2.3 via a forcing interpretation. The type system contains standard types such as dependent products of the form  $\prod x:t_1.t_2$  and dependent sums of the form  $\sum x:t_1.t_2$ . We write  $t_1 \rightarrow t_2$  for the non-dependent  $\prod$  type; **Unit** for  $\underline{0}=\underline{0} \in \text{Nat}$ ; **Void** for  $\underline{0}=\underline{1} \in \text{Nat}$ ;  $\neg T$  for  $(T \rightarrow \text{Void})$ ; and **Bool** for  $\text{Unit} + \text{Unit}$ .

To capture the time progression notion which underlines choice operators,  $\text{TT}_{\mathcal{C}}^{\square}$  is parameterized by a Kripke frame [26, 27], consisting of a set of *worlds*  $\mathcal{W}$  equipped with a reflexive and transitive binary relation  $\sqsubseteq$ . Let  $w$  range over  $\mathcal{W}$ . We sometimes write  $w' \sqsupseteq w$  for  $w \sqsubseteq w'$ . Let  $\mathcal{P}_w$  be the collection of predicates on world extensions, i.e., functions in  $\forall w' \sqsupseteq w. \mathbb{P}$ . Due to  $\sqsubseteq$ 's transitivity, if  $P \in \mathcal{P}_w$  then for every  $w' \sqsupseteq w$  it naturally extends to a predicate in  $\mathcal{P}_{w'}$ . Let  $\forall_w^{\sqsubseteq}(P)$  stand for the fact that  $P \in \mathcal{P}_w$  is true for all extensions of  $w$ , i.e.,  $P$  holds for all  $w' \sqsupseteq w$ . We sometime write  $\forall_w^{\sqsubseteq}(w'.P)$  instead of  $\forall_w^{\sqsubseteq}(\lambda w'.P)$ .

Fig. 1's lower part presents  $\text{TT}_{\mathcal{C}}^{\square}$ 's small-step call-by-name operational semantics, where  $t_1 \xrightarrow{w_1 \mapsto w_2} t_2$  expresses that  $t_1$  reduces to  $t_2$  in one step of computation from the world  $w_1$  and potentially updating it so that the resulting world is  $w_2$ . We omit the congruence rules such as: if  $t_1 \xrightarrow{w_1 \mapsto w_2} t_2$  then  $t_1(u) \xrightarrow{w_1 \mapsto w_2} t_2(u)$ . We denote by  $\mapsto^*$  the reflexive transitive closure of  $\mapsto$ , i.e.,  $a \xrightarrow{w_1 \mapsto w_2}^* b$  states that  $a$  computes to  $b$  in 0 or more steps. We write  $a \xrightarrow{w}^* b$  for  $\exists(w' : \mathcal{W}). a \xrightarrow{w \mapsto w'}^* b$ , and  $a \Rightarrow_w b$  for  $\forall_w^{\sqsubseteq}(w'. a \xrightarrow{w'}^* b)$ .

$\text{TT}_{\mathcal{C}}^{\square}$  includes effectful notions that rely on worlds to record choices and provides operators to access and update choices. In this paper, for conciseness of presentation, we focus on one instance of choice operators as mutable references to natural numbers. Reference cells, which allow a program to indirectly access a particular object, are choice operators since they can point to different objects over their lifetime. See [10] for the general notion of choice operators. To define references to numbers<sup>2</sup>, we let the set of choices  $\mathcal{C} \subseteq \text{Term}$  to be  $\mathbb{N}$ . A choice stored in a reference cell is referred to through the reference's name. To this end,  $\text{TT}_{\mathcal{C}}^{\square}$ 's computation system is parameterized by a set  $\mathcal{N}$  of choice names, ranged over by  $\delta$ , equipped with a decidable equality, and an operator that given a list of names, returns a name not in the list ( $\mathcal{N} \equiv \mathbb{N}$  for simplicity). This can be given by nominal sets [30]. We take worlds to be lists of cells, where a cell is a pair of a choice name and a choice, and  $\sqsubseteq$  is the reflexive transitive closure of two operations that allow creating and updating reference cells.

As shown in Fig. 1, a choice name  $\delta$  can be used in a computation to access choices from a world using  $! \delta \xrightarrow{w \mapsto w} \text{read}(w, \delta)$ , where the partial function  $\text{read} \in \mathcal{W} \rightarrow \mathcal{N} \rightarrow \mathcal{C}$  accesses the content of the  $\delta$ -cell in  $w$  if that cell exists.<sup>3</sup> Choices can be made using  $(\delta := t) \xrightarrow{w \mapsto \text{write}(w, \delta, t)} \star$ , where  $\text{write}(w, \delta, t)$  updates the reference  $\delta$  with the choice  $t$  if  $\delta$  occurs in  $w$ , and otherwise returns  $w$ , and therefore  $w \sqsubseteq \text{write}(w, \delta, t)$ . The computation returns  $\star$ , which is reminiscent of reference updates in OCaml for example, which are of type **unit**. Finally, new choice names can be generated using  $\nu x.t \xrightarrow{w \mapsto \text{start}\nu\mathcal{C}(w)} t[x \setminus \nu\mathcal{C}(w)]$ , where  $\nu\mathcal{C}(w)$  returns a “fresh” name not occurring in the list  $w$ , which  $x$  gets replaced with in the expression above, and  $\text{start}\nu\mathcal{C}(w)$  returns the list  $w$  extended with the pair  $\langle \nu\mathcal{C}(w), 0 \rangle$ , where 0 is the default value with which reference cells are filled, and therefore  $\forall(w : \mathcal{W}). w \sqsubseteq \text{start}\nu\mathcal{C}(w)$ .<sup>4</sup>

<sup>2</sup> Only relevant components of the choice operator are discussed. See `worldInstanceRef.lagda` for details.

<sup>3</sup> In general, `read`,  `$\nu\mathcal{C}$` , `start $\nu\mathcal{C}$` , and `write` are all parameters of  $\text{TT}_{\mathcal{C}}^{\square}$ , as described in [10]. Here they too are instantiated with references to numbers.

<sup>4</sup>  $\text{TT}_{\mathcal{C}}^{\square}$  also contains a quotienting type operator  $\zeta$  used to assign types to computations that can compute to different values in different worlds, such as choices `!  $\delta$`  [11]. For readability, we elide it here.

- Numbers:**  $w \vDash \text{Nat} \equiv \text{Nat} \iff \text{True}$   
 $w \vDash t \equiv t' \in \text{Nat} \iff \Box_w(w'.\exists(n : \mathbb{N}).t \mapsto_{w'} n \wedge t' \mapsto_{w'} n)$
- Products:**  $w \vDash \prod x:A_1.B_1 \equiv \prod x:A_2.B_2 \iff \text{Fam}_w(A_1, A_2, \lambda x.B_1, \lambda x.B_2)$   
 $w \vDash f \equiv g \in \prod x:A.B \iff \Box_w(w'.\forall(a_1, a_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A \rightarrow w' \vDash f a_1 \equiv g a_2 \in B[x \setminus a_1])$
- Sums:**  $w \vDash \sum x:A_1.B_1 \equiv \sum x:A_2.B_2 \iff \text{Fam}_w(A_1, A_2, \lambda x.B_1, \lambda x.B_2)$   
 $w \vDash p_1 \equiv p_2 \in \sum x:A.B \iff \Box_w(w'.\exists(a_1, a_2, b_1, b_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge w' \vDash b_1 \equiv b_2 \in B[x \setminus a_1] \wedge p_1 \mapsto_{w'} \langle a_1, b_1 \rangle \wedge p_2 \mapsto_{w'} \langle a_2, b_2 \rangle)$
- Sets:**  $w \vDash \{x : A_1 \mid B_1\} \equiv \{x : A_2 \mid B_2\} \iff \text{Fam}_w(A_1, A_2, \lambda x.B_1, \lambda x.B_2)$   
 $w \vDash a_1 \equiv a_2 \in \{x : A \mid B\} \iff \Box_w(w'.\exists(b_1, b_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge w' \vDash b_1 \equiv b_2 \in B[x \setminus a_1])$
- Disjoint unions:**  $w \vDash A_1 + B_1 \equiv A_2 + B_2 \iff w \vDash A_1 \equiv A_2 \wedge w \vDash B_1 \equiv B_2$   
 $w \vDash a_1 \equiv a_2 \in A + B \iff \Box_w(w'.\exists(u, v : \text{Term}).(a_1 \mapsto_{w'} \text{inl}(u) \wedge a_2 \mapsto_{w'} \text{inl}(v) \wedge w' \vDash u \equiv v \in A) \vee (a_1 \mapsto_{w'} \text{inr}(u) \wedge a_2 \mapsto_{w'} \text{inr}(v) \wedge w' \vDash u \equiv v \in B))$
- Equalities:**  $w \vDash (a_1 = b_1 \in A) \equiv (a_2 = b_2 \in B) \iff w \vDash A \equiv B \wedge w \vDash a_1 \equiv a_2 \in A \wedge w \vDash b_1 \equiv b_2 \in B$   
 $w \vDash a_1 \equiv a_2 \in (a = b \in A) \iff \Box_w(w'.w' \vDash a \equiv b \in A)$
- Subsingletons:**  $w \vDash \|A\| \equiv \|B\| \iff w \vDash A \equiv B$   
 $w \vDash a \equiv b \in \|A\| \iff \Box_w(w'.w' \vDash a \equiv a \in A \wedge w' \vDash b \equiv b \in A)$
- Purity:**  $w \vDash \text{pure} \equiv \text{pure} \iff \top$   
 $w \vDash a_1 \equiv a_2 \in \text{pure} \iff \text{namefree}(a_1) \wedge \text{namefree}(a_2)$
- Binary intersections:**  $w \vDash A_1 \cap B_1 \equiv A_2 \cap B_2 \iff w \vDash A_1 \equiv A_2 \wedge w \vDash B_1 \equiv B_2$   
 $w \vDash a_1 \equiv a_2 \in A \cap B \iff \Box_w(w'.w' \vDash a_1 \equiv a_2 \in A \wedge w' \vDash a_1 \equiv a_2 \in B)$
- Modality closure:**  $w \vDash T_1 \equiv T_2 \iff \Box_w(w'.\exists(T'_1, T'_2 : \text{Term}).T_1 \mapsto_{w'} T'_1 \wedge T_2 \mapsto_{w'} T'_2 \wedge w' \vDash T'_1 \equiv T'_2)$   
 $w \vDash t_1 \equiv t_2 \in T \iff \Box_w(w'.\exists(T' : \text{Term}).T \mapsto_{w'} T' \wedge w' \vDash t_1 \equiv t_2 \in T')$

■ **Figure 2** Forcing Interpretation.

### 2.3 Forcing Interpretation

$\text{TT}_C^\square$ 's semantics is similar to the one presented in [10], which we recall and extend in Fig. 2. Types are interpreted via a forcing interpretation defined using induction-recursion [16] as follows, where the forcing conditions are worlds: (1) the inductive relation  $w \vDash T_1 \equiv T_2$  expresses type equality in the world  $w$ ; (2) the recursive function  $w \vDash t_1 \equiv t_2 \in T$  expresses equality in a type. We also define  $a \mapsto_{!w} b$  as  $\forall_w^\square(w'.a \mapsto_{w'}^* b)$ , capturing the fact that the computation can read using  $!\delta$  but not write, and therefore does not change the initial world (this is used in Thm. 1). Fig. 2 defines in particular the semantics of **pure**, which is inhabited by name-free terms, where  $\text{namefree}(t)$  is defined recursively over  $t$  and returns false iff  $t$  contains a choice name  $\delta$  or a fresh operator of the form  $\nu x.t$ . We also write  $\text{Fam}_w(A_1, A_2, B_1, B_2)$  for  $w \vDash A_1 \equiv A_2 \wedge \forall_w^\square(w'.\forall(a_1, a_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A_1 \rightarrow w' \vDash B_1(a_1) \equiv B_2(a_2))$ . This forcing interpretation is parameterized by a family of abstract modalities  $\Box$ , which we sometimes refer to simply as a modality, which is a function that takes a world  $w$  to its modality  $\Box_w \in \mathcal{P}_w \rightarrow \mathbb{P}$ . We often write  $\Box_w(w'.P)$  for  $\Box_w \lambda w'.P$ . To guarantee that this interpretation yields a type system in the sense of Thm. 1, we require that the modalities satisfy certain properties detailed in [10] and reminiscent of standard modal axiom schemata [14].

► **Theorem 1** ([10]).  $\text{TT}_C^\square$  is a standard type system in the sense that its forcing interpretation induced by  $\Box$  satisfies the following properties (free variables are universally quantified):

<i>transitivity:</i>	$w \vDash T_1 \equiv T_2 \rightarrow w \vDash T_2 \equiv T_3 \rightarrow w \vDash T_1 \equiv T_3$	$w \vDash t_1 \equiv t_2 \in T \rightarrow w \vDash t_2 \equiv t_3 \in T \rightarrow w \vDash t_1 \equiv t_3 \in T$
<i>symmetry:</i>	$w \vDash T_1 \equiv T_2 \rightarrow w \vDash T_2 \equiv T_1$	$w \vDash t_1 \equiv t_2 \in T \rightarrow w \vDash t_2 \equiv t_1 \in T$
<i>computation:</i>	$w \vDash T \equiv T' \rightarrow T \mapsto_{!w} T' \rightarrow w \vDash T \equiv T'$	$w \vDash t \equiv t' \in T \rightarrow t \mapsto_{!w} t' \rightarrow w \vDash t \equiv t' \in T$
<i>monotonicity:</i>	$w \vDash T_1 \equiv T_2 \rightarrow w \sqsubseteq w' \rightarrow w' \vDash T_1 \equiv T_2$	$w \vDash t_1 \equiv t_2 \in T \rightarrow w \sqsubseteq w' \rightarrow w' \vDash t_1 \equiv t_2 \in T$
<i>locality:</i>	$\Box_w(w'.w' \vDash T_1 \equiv T_2) \rightarrow w \vDash T_1 \equiv T_2$	$\Box_w(w'.w' \vDash t_1 \equiv t_2 \in T) \rightarrow w \vDash t_1 \equiv t_2 \in T$
<i>consistency:</i>	$\neg w \vDash t \equiv t \in \text{Void}$	

Note that due to effects, types are not closed under all computations. For example, when  $T \equiv \text{Nat}$ ,  $t' \Vdash_w \underline{n}$  does not necessarily follow from  $t \Vdash_w t'$  and  $t \Vdash_w \underline{n}$ . An example is  $t \equiv (\delta := \underline{1}; \text{if } !\delta < \underline{1} \text{ then } \underline{0} \text{ else } \underline{1})$ , which reduces to  $t' \equiv (\text{if } !\delta < \underline{1} \text{ then } \underline{0} \text{ else } \underline{1})$  and also to  $\underline{1}$  in all worlds, but  $t'$  does not reduce to  $\underline{1}$  in all worlds, because  $\delta$  could be initialized differently in different worlds. However, the following holds by transitivity of  $\Vdash_w$ :  $t' \Vdash_w t \rightarrow w \Vdash t \equiv t \in \text{Nat} \rightarrow w \Vdash t \equiv t' \in \text{Nat}$ . Similarly, the following also holds by transitivity of  $\Vdash_w$ :  $w \Vdash T \equiv T \rightarrow T' \Vdash_w T \rightarrow w \Vdash T \equiv T'$ . Finally, note that, as indicated in Thm. 1, this semantics is closed under  $\beta$ -reduction, as  $\beta$ -reduction does not modify the current world.

## 2.4 $\text{TT}_C^\square$ 's Inference Rules

$\text{TT}_C^\square$ 's inference rules are standard and they reflect the semantics of the types, which is given meaning through a forcing interpretation presented in Sec. 2.3. Concetely, sequents in  $\text{TT}_C^\square$  are of the form  $h_1, \dots, h_n \vdash t : T$ . Such a sequent denotes that, assuming  $h_1, \dots, h_n$ ,  $T$  is a type inhabited by  $t$ . An hypothesis  $h$  is of the form  $x:A$ , where the variable  $x$  stands for the name of the hypothesis and  $A$  its type. We write  $a \in A$  for  $a = a \in A$ . To illustrate the naturality of the typing rules and their correspondence to the forcing interpretation, we provide examples of  $\text{TT}_C^\square$ 's inference rules for  $\Pi$  types. The following rules are the standard  $\Pi$ -elimination,  $\Pi$ -introduction, type equality for  $\Pi$  types, and  $\lambda$ -introduction rules, respectively.

$$\frac{H, f: \Pi x:A. B, J \vdash a \in A \quad H, f: \Pi x:A. B, J, z: f(a) \in B[x \setminus a] \vdash e : C}{H, f: \Pi x:A. B, J \vdash e[z \setminus *] : C} \quad \frac{H, z:A \vdash b : B[x \setminus z] \quad H \vdash A \in \mathbb{U}_i}{H \vdash \lambda z. b : \Pi x:A. B}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y:A_1 \vdash B_1[x_1 \setminus y] = B_2[x_2 \setminus y] \in \mathbb{U}_i}{H \vdash \Pi x_1:A_1. B_1 = \Pi x_2:A_2. B_2 \in \mathbb{U}_i} \quad \frac{H, z:A \vdash t_1[x_1 \setminus z] = t_2[x_2 \setminus z] \in B[x \setminus z] \quad H \vdash A \in \mathbb{U}_i}{H \vdash \lambda x_1. t_1 = \lambda x_2. t_2 \in \Pi x:A. B}$$

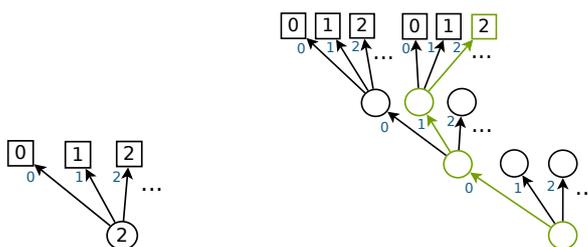
The following rules are the standard function extensionality and  $\beta$ -reduction rules, resp.:

$$\frac{H, z:A \vdash f_1(z) = f_2(z) \in B[x \setminus z] \quad H \vdash A \in \mathbb{U}_i}{H \vdash f_1 = f_2 \in \Pi x:A. B} \quad \frac{H \vdash t[x \setminus s] = u \in T}{H \vdash (\lambda x. t) s = u \in T}$$

## 3 Inductive Continuity via Brouwer Trees

This section states a dialogue tree-based continuity principle, referred to as the inductive continuity principle, since it relies on trees to capture functions. As we show in Sec. 4, it implies both Brouwer's continuity principle for numbers and his uniform continuity principle on the Cantor space. Furthermore, it is still unknown whether the inductive continuity principle is strictly stronger than Brouwer's continuity principle for numbers. Sec. 5 internally validates this inductive principle. In particular, Thm. 4 shows that, given a pure function  $F \in \mathfrak{B} \rightarrow \text{Nat}$ ,  $\text{TT}_C^\square$  provides a computation, introduced in Sec. 5.1, that builds a dialogue tree capturing  $F$ 's continuity.

As mentioned above, we rely here on Brouwer trees, which are a simple form of dialogue trees. Let us provide an example of how dialogue and Brouwer trees work. Consider the function  $F \equiv \lambda \alpha. \alpha(2) \in \mathfrak{B} \rightarrow \text{Nat}$ . Fig. 3 (left) shows its dialogue tree, where the internal (root) node is labeled with the value  $\alpha$  is applied to, and the leaves contain the values of  $F$  for all possible inputs. For example if  $F$  is applied to  $\alpha \equiv \lambda x. x$ , then starting from the root, we apply  $\alpha$  to the node's value, i.e.,  $\underline{2}$ , which gives us  $\underline{2}$ , and we therefore follow the 2nd path, which leads to the leaf labeled 2, the value of  $F(\alpha)$ . If  $\alpha \equiv \lambda x. \underline{0}$ , then  $\alpha(2)$  is now  $\underline{0}$ , and following the 0th path leads to the leaf labeled 0, which is the value of  $F(\alpha)$ . Fig. 3 (right) shows  $F$ 's Brouwer tree, where as opposed to dialogue trees, internal nodes are not labeled, and as for dialogue trees, the leaves contain the values of  $F$  for all inputs. For



■ **Figure 3** Examples of dialogue (left) and Brouwer (right) trees for  $\lambda\alpha.\alpha(2)$ .

example if  $F$  is applied to  $\alpha \equiv \lambda x.x$ , because  $\alpha(0)$  is  $\underline{0}$ , we first follow the 0th branch; then because  $\alpha(1)$  is  $\underline{1}$ , we follow the 1st branch, and finally because  $\alpha(2)$  is  $\underline{2}$ , we follow the 2nd branch, leading to a leaf labeled 2 (following the green path in Fig. 3). If  $\alpha \equiv \lambda x.\underline{0}$ , then we instead always follow the 0th branch, leading to a leaf labeled with 0.

In the dialogue tree, the modulus of continuity of  $F$  at some point  $\alpha$  is given by the maximum value of the internal nodes followed using  $\alpha$ , while in the Brouwer tree, the modulus is the length of the branch followed using  $\alpha$ . Note that, in general, the values of the internal nodes of a dialogue tree of a function  $F \in \mathfrak{B} \rightarrow \text{Nat}$  are used to “ask questions” to an argument  $\alpha \in \mathfrak{B}$  to decide what branch to take in the tree (by applying  $\alpha$  to those values), while in a Brouwer tree, “dialogues” happen by asking all the values of an initial segment of  $\alpha$ .

### 3.1 Extending $\text{TT}_{\mathcal{C}}^{\square}$ with (Co-)W Types and Infinite Sequences

In order to state the inductive continuity principle, we make use of the notion of a Brouwer tree, which we define in  $\text{TT}_{\mathcal{C}}^{\square}$  using W types [1, Sec.5.2], which is a standard way of representing inductive types. Additionally, we use co-W types (also called M types) [1, Sec.5.2], the dual notion to that of a W type, to prove the validity of the principle. Thus, we add W and M types to  $\text{TT}_{\mathcal{C}}^{\square}$ , using  $\text{sup}$  as a W type and M type constructor and  $\text{wrec}$  as a W type recursor.

$$\begin{aligned}
 vt \in \text{Type} & ::= \dots \mid W(t_1, t_2) \mid M(t_1, t_2) \\
 t \in \text{Term} & ::= \dots \mid \text{sup}(t_1, t_2) \mid \text{wrec}(t_1, t_2) \\
 v \in \text{Value} & ::= \dots \mid \lceil \mathbf{s} \rceil, \text{ where } \mathbf{s} \text{ is a metatheoretical function in } \mathbb{N} \rightarrow \mathbb{N}
 \end{aligned}$$

where  $\text{wrec}(t_1, t_2)$  and  $\lceil \mathbf{s} \rceil$  compute as follows:

$$\text{wrec}(\text{sup}(a, f), g) \quad w \mapsto_w \quad g \ a \ f \ (\lambda b. \text{wrec}(f(b), g)) \quad \lceil \mathbf{s} \rceil \ n \quad w \mapsto_w \quad \underline{\mathbf{s}(n)}$$

In addition, the application operator is modified so that it evaluates its argument whenever the function is of the form  $\lceil \mathbf{s} \rceil$ , i.e.,  $\lceil \mathbf{s} \rceil \ a$  reduces to  $\lceil \mathbf{s} \rceil \ b$  when  $a$  reduces to  $b$ . Hence, for any metatheoretical function  $\mathbf{s}$  in  $\mathbb{N} \rightarrow \mathbb{N}$ ,  $\lceil \mathbf{s} \rceil$  inhabits  $\mathfrak{B}$ . These sequences are used in Sec. 5.5 to prove that the computation of Brouwer trees provided in Sec. 5.1 terminates. They are similar to the sequences of the form  $\lambda x.M_x$  in [5], where the infinite sequence of terms  $M_1, M_2, \dots$  does not have a computational purpose, but is used to prove termination in their proof that some bar recursion operator realizes the negative translation of the axiom of choice. Similar sequences have been used in [31] to validate versions of the axiom of choice, and in [34] to validate variants of Brouwer’s Bar Induction principle [24].

W and M types are interpreted in a standard way:

**W types:**  $w \vDash W(A_1, B_1) \equiv W(A_2, B_2) \iff \text{Fam}_w(A_1, A_2, B_1, B_2)$   
 $w \vDash s_1 \equiv s_2 \in W(A, B) \iff \Box_w(w'.\mu(R.\exists(a_1, a_2, f_1, f_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge$   
 $(\forall(b_1, b_2 : \text{Term}).w' \vDash b_1 \equiv b_2 \in B(a_1) \rightarrow R f_1(b_1) f_2(b_2)) \wedge s_1 \vDash_{w'} \text{sup}(a_1, f_1) \wedge s_2 \vDash_{w'}$   
 $\text{sup}(a_2, f_2)) s_1 s_2)$

**M types:**  $w \vDash M(A_1, B_1) \equiv M(A_2, B_2) \iff \text{Fam}_w(A_1, A_2, B_1, B_2)$   
 $w \vDash s_1 \equiv s_2 \in M(A, B) \iff \Box_w(w'.\nu(R.\exists(a_1, a_2, f_1, f_2 : \text{Term}).w' \vDash a_1 \equiv a_2 \in A \wedge$   
 $(\forall(b_1, b_2 : \text{Term}).w' \vDash b_1 \equiv b_2 \in B(a_1) \rightarrow R f_1(b_1) f_2(b_2)) \wedge s_1 \vDash_{w'} \text{sup}(a_1, f_1) \wedge s_2 \vDash_{w'}$   
 $\text{sup}(a_2, f_2)) s_1 s_2)$

Therefore,  $W(A, B)$  and  $M(A, B)$  are types in  $\mathbb{U}_i$  whenever  $A \in \mathbb{U}_i$  and  $B \in A \rightarrow \mathbb{U}_i$ . Given  $a \in A$  and  $f \in B[a] \rightarrow W(A, B)$ ,  $\text{sup}(a, f) \in W(A, B)$  is a W type constructor, and if  $f \in B[a] \rightarrow M(A, B)$  then  $\text{sup}(a, f) \in M(A, B)$  is an M type constructor. Given  $t \in W(A, B)$  and  $g \in \Pi a:A.(B(a) \rightarrow W(A, B)) \rightarrow (B(a) \rightarrow C) \rightarrow C$ ,  $\text{wrec}(t, g) \in C$  is a W type recursor.

► **Example 2.** Given  $A \in \mathbb{U}_i$  and  $B \in A \rightarrow \mathbb{U}_i$ ,  $W(A, B)$  denotes the type of inductive definitions with inhabitants of  $A$  representing the constructors (as well as their non-inductive parameters), and  $B(a)$  representing the indices of inductive parameters at a given constructor  $a$ . For example, the natural numbers have two constructors: **zero** and **succ**, the latter having one inductive parameter. Therefore, natural numbers are encoded as:

$$W(\text{Bool}, \lambda x.\text{case } x \text{ of } \text{inl}(\_) \Rightarrow \text{Void} \mid \text{inr}(\_) \Rightarrow \text{Unit}),$$

where **Void** captures the lack of inductive parameters for **zero** and **Unit** captures **succ**'s single inductive parameter. The constructors **zero** and **succ** are then be encoded as:

$$\text{zero} \equiv \text{sup}(\text{inl}(\star), \lambda x.\star) \quad \text{and} \quad \text{succ} \equiv \lambda n.\text{sup}(\text{inr}(\star), \lambda x.n)$$

### 3.2 Brouwer Tree-Based Inductive Continuity Principle

We can now state the inductive continuity principle that captures the moduli of continuity of functions in  $\mathfrak{B}_{\text{SNat}} \rightarrow \text{Nat}$  using Brouwer trees, where  $\mathfrak{B}_{\text{SNat}} \equiv \text{Nat} \rightarrow \text{SNat}$  for  $\text{SNat}$  a subtype of  $\text{Nat}$  (this principle is therefore a family of principles for all such  $\text{SNats}$ ). This continuity result, as well as the ones recalled in Sec. 4, are stated for pure functions only using the following quantification:  $\Pi_p a:A.B \equiv \Pi a:(A \cap \text{pure}).B$ , which quantifies over pure members of  $A$ . We also write  $A_p$  for  $A \cap \text{pure}$  and  $A +_p B$  for  $(A+B) \cap \text{pure}$ . It remains to be determined whether some effectful computations can be proved to be continuous.

We first define Brouwer trees (a class of dialogue trees where internal nodes are not labeled) using W types as follows.

► **Definition 3** (Brouwer Trees). *A Brouwer tree is a member of  $\text{Bt} \equiv W(\text{BtA}, \text{BtB})$ , where  $\text{BtA} \equiv \text{Nat} +_p \text{Unit}$  and  $\text{BtB} \equiv \lambda a.\text{if } a \text{ then } \text{Void} \text{ else } \text{SNat}_p$ . Such trees have two constructors:  $\eta(i) \equiv \text{sup}(\text{inl}(i), \lambda x.\star)$ , which builds a leaf node with value  $i \in \text{Nat}$ ; and  $F(f) \equiv \text{sup}(\text{inr}(\star), f)$ , which builds an internal node from a function  $f \in \text{SNat}_p \rightarrow \text{Bt}$ .*

Using this definition, the Brouwer tree depicted in Fig. 3 is  $F(\lambda i.F(\lambda j.F(\lambda k.\eta(k))))$ .

► **Theorem 4** (Inductive Continuity Principle). *The following continuity principle, referred to as  $\text{ICP}_p$ , is valid in  $\text{TT}_C^{\square 5}$  (see `contDiagVal` in `barContP10.lagda` for details):*

$$\Pi_p F:\mathfrak{B}_{\text{SNat}} \rightarrow \text{Nat}.\|\Sigma d:\text{Bt}.\Pi_p \alpha:\mathfrak{B}_{\text{SNat}}.\text{follow}(d, \alpha) = F(\alpha) \in \text{Nat}\| \quad (\text{ICP}_p)$$

where  $\text{follow}(d, \alpha)$  extracts the value of the leaf encountered when following  $\alpha$  in  $d$  as follows:

$$\text{follow}(d, \alpha) \equiv \text{wrec}(d, \lambda a.\lambda f.\lambda r.\lambda k.\text{case } a \text{ of } \text{inl}(i) \Rightarrow i \mid \text{inr}(\_) \Rightarrow r (\alpha k) (k+1)) \ 0$$

<sup>5</sup> “Valid in  $\text{TT}_C^{\square}$ ” here means that the principle is realizable in  $\text{TT}_C^{\square}$ , thus it is consistent with the theory.

At a high-level, the proof goes as follows (the full proof is carried out in Sec. 5).

- Step 1:** Given a function in  $\mathfrak{B}_{\text{SNat}} \rightarrow \text{Nat}$ , we first build by coinduction a possibly infinite co-Brouwer tree as an M type. This co-Brouwer tree contains the result of  $F$  applied to the finite sequence  $s$  at the leaf ending the path following  $s$  whenever  $s$  contains enough information to compute the result of  $F$ .
- Step 2:** Classically, this co-Brouwer tree is either finite or contains an infinite branch.
- Step 3:** If the co-Brouwer tree is finite, it is a Brouwer tree.
- Step 4:** If the co-Brouwer tree contains an infinite branch, then the branch gives rise to an infinite sequence  $\alpha$ , and since  $F$  is continuous, the path must be finite. As discussed in Sec. 5.5, this step relies on a continuity argument similar to the one used to validate the weak continuity principle  $\text{WCP}_p$  recalled in Sec. 4.1.
- Step 5:** Finally, the obtained Brouwer tree is shown to contain the values of  $F$  at its leaves.

## 4 Relation with Other Continuity Principles

This section demonstrates that inductive continuity implies both Brouwer’s continuity principle for numbers (referred to as weak continuity here) and uniform continuity.

### 4.1 Weak Continuity

$\text{TT}_C^\square$  was shown to satisfy the following version of Brouwer’s continuity principle for numbers, also called the weak continuity principle, which therefore can be added as an axiom [11].

$$\prod_p F : \mathfrak{B} \rightarrow \text{Nat} . \prod_p \alpha : \mathfrak{B} . \|\Sigma n : \text{Nat} . \prod_p \beta : \mathfrak{B} . (\alpha = \beta \in \mathfrak{B}_n) \rightarrow (F(\alpha) = F(\beta) \in \text{Nat})\| \quad (\text{WCP}_p)$$

$\text{WCP}_p$  is realized in every world by the term  $\lambda F . \lambda \alpha . \langle \text{mod}(F, \alpha), \lambda \beta . \lambda e . \star \rangle$ , where  $\text{mod}(F, \alpha)$  computes the modulus of continuity of the function  $F \in \mathfrak{B} \rightarrow \text{Nat}$  at  $\alpha \in \mathfrak{B}$ . Roughly speaking,  $\text{mod}(F, \alpha)$  generates a reference cell  $\delta$  initialized with 0, applies  $F$  to a modified version of  $\alpha$  (namely  $\text{upd}(\delta, \alpha)$ ) that keeps track using  $\delta$  of the highest number  $\alpha$  gets applied to, and then returns the value held by  $\delta$  (plus one). Formally:

$$\begin{aligned} \text{mod}(F, \alpha) &\equiv \nu x . (x := \underline{0}; F(\text{upd}(x, \alpha)); !x + \underline{1}) \\ \text{upd}(\delta, \alpha) &\equiv \lambda x . (\text{let } y = x \text{ in } ((\text{if } !\delta < y \text{ then } \delta := y \text{ else } \star); \alpha(y))) \end{aligned}$$

Note that the truncation in  $\text{WCP}_p$  is necessary. It has been shown that a non-truncated version of  $\text{WCP}$  is inconsistent with MLTT [18, 40], and the same applies to  $\text{WCP}_p$  and  $\text{TT}_C^\square$ . The main reason for this is the semantics of dependent functions given by  $\text{TT}_C^\square$ ’s realizability model (see Fig. 2). Under this semantics,  $f \in \prod x : A . B$  if  $f$  maps equal terms  $a_1 = a_2 \in A$  to equal terms  $f(a_1) = f(a_2) \in B[x \setminus a_1]$ . As continuity is a non-extensional property [25], extensionally equal functions in  $\mathfrak{B}$  might have different moduli of continuity, so  $\text{WCP}_p$ ’s realizer cannot inhabit a non-truncated version of  $\text{WCP}_p$ . However, when  $B$  is of the form  $\|C\|$ , it suffices that  $f(a_1)$  and  $f(a_2)$  are both members of  $C[x \setminus a_1]$ , allowing  $\text{WCP}_p$ ’s validation.

► **Theorem 5.**  $\text{WCP}_p$  is derivable from  $\text{ICP}_p$  in  $\text{TT}_C^\square$  when  $\text{SNat} \equiv \text{Nat}$ .

**Proof outline.** Let  $F \in \mathfrak{B} \rightarrow \text{Nat}$  a pure function and let  $\alpha \in \mathfrak{B}$ . It follows from  $\text{ICP}_p$  that:  $\|\Sigma d : \text{Bt} . \prod_p \alpha : \mathfrak{B} . \text{follow}(d, \alpha) = F(\alpha) \in \text{Nat}\|$ . Because both principles are truncated, we can assume the existence of a tree  $d \in \text{Bt}$  such that:  $\prod_p \alpha : \mathfrak{B} . \text{follow}(d, \alpha) = F(\alpha) \in \text{Nat}$ . Because  $d$  encodes the modulus of continuity of each sequence  $\alpha \in \mathfrak{B}$ , as the length of the branch in  $d$  that “follows”  $\alpha$ , we instantiate the conclusion with:  $n \equiv \text{lenBranch}(d, \alpha) \in \text{Nat}$ , where:

$$\text{lenBranch}(d, \alpha) \equiv \text{wrec}(d, \lambda a . \lambda f . \lambda r . \lambda k . \text{case } a \text{ of } \text{inl}(i) \Rightarrow k \mid \text{inr}(\_) \Rightarrow r(\alpha \ k) \ (k + \underline{1}) \ \underline{0}$$

## 37:10 Inductive Continuity via Brouwer Trees

It now remains to prove that  $F(\alpha)=F(\beta)\in\mathbf{Nat}$ , for any pure function  $\beta \in \mathfrak{B}$  such that  $\alpha=\beta\in\mathfrak{B}_n$ . From  $\mathbf{ICP}_p$ , we know that  $\mathbf{follow}(d, \alpha)=F(\alpha)\in\mathbf{Nat}$  and  $\mathbf{follow}(d, \beta)=F(\beta)\in\mathbf{Nat}$ . Therefore, it is enough to prove  $\mathbf{follow}(d, \alpha)=\mathbf{follow}(d, \beta)\in\mathbf{Nat}$ , which follows from the following fact:  $\prod\alpha, \beta:\mathfrak{B}.\alpha=\beta\in\mathfrak{B}_{\mathbf{lenBranch}(d, \alpha)} \rightarrow \mathbf{follow}(d, \alpha)=\mathbf{follow}(d, \beta)\in\mathbf{Nat}$ . ◀

### 4.2 Uniform Continuity

The uniform continuity principle states that all functions on the Cantor space ( $\mathfrak{C} \equiv \mathbf{Nat} \rightarrow \mathbf{Bool}$ ) are uniformly continuous, meaning that all points  $\alpha \in \mathfrak{C}$  have the same modulus of continuity. We consider here the following version:

$$\prod_p F:\mathfrak{C} \rightarrow \mathbf{Nat}.\|\sum n:\mathbf{Nat}.\prod_p \alpha, \beta:\mathfrak{C}.\alpha=\beta\in\mathfrak{C}_n \rightarrow (F(\alpha)=F(\beta)\in\mathbf{Nat})\| \quad (\mathbf{UCP}_p)$$

Brouwer proved that all real-valued functions on the unit interval are uniformly continuous [8, Thm.3] using  $\mathbf{WCP}$  and the Fan Theorem [36, 15], which he derived from Bar Induction. While it was shown that in the case of uniform continuity the truncation can be removed [18, 40], we leave formalizing this in  $\mathbf{TT}_{\mathfrak{C}}^{\square}$  for future work.

► **Theorem 6.**  $\mathbf{UCP}_p$  is derivable from  $\mathbf{ICP}_p$  in  $\mathbf{TT}_{\mathfrak{C}}^{\square}$  when  $\mathbf{SNat} \equiv \{x : \mathbf{Nat} \mid x < 2\}$  or equivalently  $\mathbf{SNat} \equiv \mathbf{Bool}$  (and therefore  $\mathfrak{B}_{\mathbf{SNat}}$  is  $\mathfrak{C}$ ).

**Proof outline.** Let  $F \in \mathfrak{C} \rightarrow \mathbf{Nat}$  be a pure function. Because both principles are truncated, we can assume the existence of a tree  $d \in \mathbf{Bt}$  such that:  $\prod_p \alpha:\mathfrak{C}.\mathbf{follow}(d, \alpha)=F(\alpha)\in\mathbf{Nat}$ . As  $d$  is finitely branching and encodes the modulus of continuity of each  $\alpha \in \mathfrak{C}$  as the length of the branch in  $d$  that “follows”  $\alpha$ , we compute the uniform modulus of continuity of  $F$  as  $d$ ’s depth as follows, where  $\mathbf{max}(i, j)$  returns the maximum among the numbers  $i$  and  $j$ :

$$\mathbf{depth}(d) \equiv \mathbf{wrec}(d, \lambda a.\lambda f.\lambda r.\mathbf{case} \ a \ \mathbf{of} \ \mathbf{inl}(i) \Rightarrow \underline{1} \mid \mathbf{inr}(\_) \Rightarrow \mathbf{max}(r(0), r(1)) + 1)$$

We then instantiate our conclusion with  $n \equiv \mathbf{depth}(d) \in \mathbf{Nat}$ , and have to prove that  $F(\alpha)=F(\beta)\in\mathbf{Nat}$ , for all pure functions  $\alpha, \beta \in \mathfrak{C}$  such that  $\alpha=\beta\in\mathfrak{C}_n$ . From  $\mathbf{ICP}_p$ , we know that  $\mathbf{follow}(d, \alpha)=F(\alpha)\in\mathbf{Nat}$  and  $\mathbf{follow}(d, \beta)=F(\beta)\in\mathbf{Nat}$ . Therefore, it is enough to prove  $\mathbf{follow}(d, \alpha)=\mathbf{follow}(d, \beta)\in\mathbf{Nat}$ , which follows from the following fact, which can be proved by induction on  $d$ :  $\prod\alpha, \beta:\mathfrak{C}.\alpha=\beta\in\mathfrak{C}_{\mathbf{depth}(d)} \rightarrow \mathbf{follow}(d, \alpha)=\mathbf{follow}(d, \beta)\in\mathbf{Nat}$ . ◀

## 5 Validity of the Inductive Continuity Principle

This section sketches the proof of Thm. 4, which has been formalized in Agda. For simplicity we focus here on functions in  $\mathfrak{B} \rightarrow \mathbf{Nat}$ , but as mentioned in Sec. 3, the principle holds for all functions in  $\mathfrak{B}_{\mathbf{SNat}} \rightarrow \mathbf{Nat}$  where  $\mathbf{SNat}$  is a subtype of  $\mathbf{Nat}$ .

To validate  $\mathbf{ICP}_p$  we assume that  $\mathbf{TT}_{\mathfrak{C}}^{\square}$ ’s  $\square$  modality is a Kripke-like modality, i.e.,  $\forall(w : \mathcal{W}).\square_w f \rightarrow \forall_w^{\square}(f)$ . This is used to derive a co-Brouwer tree from an  $F \in \mathfrak{B} \rightarrow \mathbf{Nat}$ . In short, when building a co-Brouwer tree in **Step 1** by extending a node with branches for all  $n \in \mathbf{Nat}$ , if  $n$  does not compute to a number in the current world  $w$  (which a Kripke modality enforces), it is unclear how this can result in a co-tree in  $w$ . It was proved in [10] that  $\mathbf{TT}_{\mathfrak{C}}^{\square}$  is inconsistent with classical logic when  $\square$  is a Kripke modality and  $\mathfrak{C}$  is instantiated using references, which is expected because continuity contradicts classical logic [36, 39].

## 5.1 Computing Brouwer Trees

To show that  $\text{ICP}_p$  is valid, we must exhibit a  $\text{TT}_C^\square$  computation that can compute a Brouwer tree from a pure function in  $\mathfrak{B} \rightarrow \text{Nat}$ . This computation is similar to the one provided in [35, Sec.1.3], and proceeds as follows: given  $F \in \mathfrak{B} \rightarrow \text{Nat}$ ,  $\text{loop}(F) \underline{0} \alpha_0$  builds a tree in  $\text{Bt}$  satisfying the condition in Thm. 4, where  $\alpha_0 \equiv \lambda\_.\underline{0}$ , and  $\text{loop}$  is defined as follows:

$$\begin{aligned} \text{loop}(F) &\equiv \text{fix}(\lambda R. \lambda k. \lambda \alpha. \nu x. (x := \underline{0}); \text{let } i = F(\text{upd}(x, \alpha)) \text{ in } \text{cases}(x, R, k, \alpha, i)) \\ \text{cases}(\delta, R, k, \alpha, i) &\equiv \text{if } !\delta < k \text{ then } \eta(i) \text{ else } F(\lambda x. R (k + \underline{1}) \text{ append}(k, \alpha, x)) \end{aligned}$$

The goal of this computation is to recursively build a Brouwer tree from the root, by applying  $F$  to a finite sequence (essentially, the pair  $\langle k, \alpha \rangle$ ), which corresponds to a path in the tree, and which is extended as long as it does not contain enough information for  $F$  to compute a value, i.e., as long as  $F$  makes use of more than  $k$  values from  $\alpha$ .

Note that a finite sequence, or a list, of elements of type  $A$  is encoded here as a pair of its length  $k$  and a function in  $\text{Nat} \rightarrow A$  where only its initial segment of length  $k$  is relevant. Given a list  $l$  given by the pair  $k$  and  $f$ , the operator  $\text{append}(k, f, a) \equiv \lambda x. \text{if } x = k \text{ then } a \text{ else } f(x)$  returns a list of length  $k + 1$  that appends  $a$  to  $l$ . Lists are defined like this instead of using a  $W$  type because  $\text{loop}(F)$  applies  $F$  to a function with initial segment the list given as argument. Therefore, instead of using an additional operator to turn an element of such a  $W$  type into a function, with this encoding lists directly provide such functions.

The computation in [35] uses exceptions to test whether  $F$  requires more values than the ones provided in the current finite sequence, while we use here references as in [11]. Exceptions are well-suited to test whether the modulus of continuity is reached, but not to directly compute moduli of continuity. For example, the computation in [32] relies on exceptions and a loop, while the computation in [11] makes use of references and does not require an additional loop because a reference cell can be used to store the moduli of continuity. Instead of using a reference to a Boolean, which would be similar to using an exception, we use here a reference  $\delta$  that points to a number, and apply  $F$  to  $\text{upd}(\delta, \alpha)$ , as in  $\text{WCP}_p$ 's realizer, as it allows us to reuse some of the results used in [11] to validate  $\text{WCP}_p$ .

## 5.2 Step 1: Building a co-W

First, we prove that from a function  $F \in \mathfrak{B} \rightarrow \text{Nat}$ , we get  $\text{loop}(F) \underline{0} \alpha_0 \in \text{CoDiag}$ , where  $\text{CoDiag} \equiv M(\text{Nat} +_p \text{Unit}, \lambda a. \text{if } a \text{ then } \text{Void} \text{ else } \text{Nat}_p)$ . We prove this by coinduction, and by inspecting the computation of  $\text{loop}(F)$  (see `coSem` in `barContP2.lagda`). Given  $k \in \text{Nat}_p$  and  $\alpha \in \mathfrak{B}$ ,  $(\text{loop}(F) k \alpha)$  first evaluates  $F(\text{upd}(\delta, \alpha))$  to  $i$  for some “fresh”  $\delta$ , and then returns  $\eta(i)$  if  $!\delta < k$ , and otherwise returns  $F(\lambda x. \text{loop}(F) (k + \underline{1}) \text{ append}(k, \alpha, x))$ . We now prove  $\text{loop}(F) k \alpha \in \text{CoDiag}$  by cases. If  $!\delta < k$  then it remains to prove that  $\eta(i) \in \text{CoDiag}$ , which is straightforward because  $F(\text{upd}(\delta, \alpha)) \in \text{Nat}$ , and therefore  $i$  too. If  $!\delta \not< k$  then it remains to prove  $F(\lambda x. \text{loop}(F) (k + \underline{1}) \text{ append}(k, \alpha, x)) \in \text{CoDiag}$ , which follows from the fact that  $\lambda x. \text{loop}(F) (k + \underline{1}) \text{ append}(k, \alpha, x) \in \text{Nat}_p \rightarrow \text{CoDiag}$ , which follows by coinduction.

## 5.3 Step 2: Case analysis

Using classical logic we analyze two cases: given  $t \in M(A, B)$ , either  $t$ 's branches are all finite or there exists an infinite branch, where the type of branches w.r.t. the world  $w$ , type  $A$ , and family  $B$  is defined as follows, a right injection capturing the termination of a branch:

$$\text{Branch} \equiv \forall (n : \mathbb{N}). (\exists (a, b : \text{Term}). w \vDash a \equiv a \in A \wedge w \vDash b \equiv b \in B(a)) \vee \top$$

## 37:12 Inductive Continuity via Brouwer Trees

Note that a branch can either be finite if it returns an element of the right disjunct (i.e.,  $\top$ ) for some  $n \in \mathbb{N}$ , or infinite if it always returns an element of the left disjunct for all  $n \in \mathbb{N}$ . Branches are defined w.r.t. a term  $t$  in  $W(A, B)$  or in  $M(A, B)$ , and we say that *a branch  $p \in \text{Branch}$  is a branch of a term  $t$*  if:  $\forall (n : \mathbb{N}). p \in_n t$ , where  $p \in_n t$  is defined recursively as follows (for  $\text{shift}(p) \equiv \lambda k.p(k+1)$ ):

$$p \in_0 t \equiv \top \quad p \in_{n+1} t \equiv \begin{cases} \exists (f : \text{Term}). t \Rightarrow_w \text{sup}(a, f) \wedge \text{shift}(p) \in_n f b, \\ \quad \text{when } p(0) \text{ is a left injection of } (a, b, \_, \_) \\ \top, \text{ otherwise} \end{cases}$$

The tree  $t \in M(A, B)$  is  $\text{loop}(F) \sqcup \alpha_0$ . In case  $t$ 's branches are all finite, we show that  $t \in W(A, B)$  (Sec. 5.4). In case  $t$  has an infinite branch, we derive a contradiction using an argument similar to one used to validate weak continuity in [11] (Sec. 5.5).

### 5.4 Step 3: Building a W type

In case  $t$ 's branches are all finite, we prove that if  $t \in M(A, B)$  then  $t \in W(A, B)$ . Again, we use classical logic: assuming  $t \notin W(A, B)$  and deriving a contradiction. Given that  $t \in M(A, B)$  and  $t \notin W(A, B)$ , we extract, by coinduction, an infinite co-branch  $u$  from  $t$ , where the type of co-branches  $u$  w.r.t. the world  $w$ , type  $A$ , and family  $B$ , is coinductively defined as follows (see `m2mb` in `barContP.lagda`):

$$\nu(R.\exists(a, f, b : \text{Term}).u \Rightarrow_w \text{sup}(a, f) \wedge w \vDash b \equiv b \in B(a) \wedge R f(b))$$

In particular, such a co-branch provides a sequence of  $B$ s. From this co-branch  $u$ , we build an infinite branch  $p \in \text{Branch}$  (see `mb2path` in `barContP.lagda`), which is a function from  $n \in \mathbb{N}$  to (left injections of)  $B$ s along with their corresponding  $A$ s, derived by induction on  $n$ . From the assumption that  $t$ 's branches are all finite we obtain that  $p$  must also be finite, from which we derive a contradiction (see `m2w` in `barContP.lagda`).

### 5.5 Step 4: Termination

In case  $t$ , which is here  $\text{loop}(F) \sqcup \alpha_0$ , contains an infinite branch  $p$ , we derive a contradiction from  $F$ 's continuity. Because  $p$  is infinite, i.e., only returns left injections, we obtain a metatheoretical function of the following type, which follows the branch  $p$  of  $\text{loop}(F) \sqcup \alpha_0$ :

$$\mathbb{N} \rightarrow \exists(a, b : \text{Term}).w \vDash a \equiv a \in \text{Bt}A \wedge w \vDash b \equiv b \in \text{Bt}B(a)$$

Therefore, for each  $n \in \mathbb{N}$ , there are two cases: either ( $w \vDash a \equiv a \in \text{Nat}$  and  $w \vDash b \equiv b \in \text{Void}$ ) or ( $w \vDash a \equiv a \in \text{Unit}$  and  $w \vDash b \equiv b \in \text{Nat}_p$ ). Since  $\text{Void}$  is not inhabited, it must be that  $w \vDash a \equiv a \in \text{Unit}$  and  $w \vDash b \equiv b \in \text{Nat}_p$ . Hence, from this function, we obtain a metatheoretical function of the following type, which follows the branch  $p$  of  $\text{loop}(F) \sqcup \alpha_0$ :

$$\mathbb{N} \rightarrow \exists(b : \text{Term}).w \vDash b \equiv b \in \text{Nat}_p$$

From this function, since  $\square$  is a Kripke-like modality, we obtain a metatheoretical function  $\mathbf{s} \in \mathbb{N} \rightarrow \mathbb{N}$ , which given  $n \in \mathbb{N}$  returns the path taken in the  $n^{\text{th}}$   $F$  along the branch  $p$  following the computation  $\text{loop}(F) \sqcup \alpha_0$ . As explained in Sec. 3.1,  $\text{TT}_{\mathcal{C}}^{\square}$ 's calculus includes all metatheoretical functions from  $\mathbb{N}$  to  $\mathbb{N}$ , which inhabit  $\mathfrak{B}$ . These sequences do not have any computational purpose here, and are only used to prove termination. We have  $\lceil \mathbf{s} \rceil \in \mathfrak{B}$ , so by continuity of  $F$  we know that there is a  $k \in \mathbb{N}$  such that the  $k^{\text{th}}$  iteration of  $\text{loop}(F) \sqcup \alpha_0$  runs  $F(\text{upd}(\delta, \lceil \mathbf{s} \rceil))$  for some “fresh”  $\delta$  such that  $\delta$ 's value stays under  $k$  during the computation

of  $F(\text{upd}(\delta, [\mathbf{s}]))$ . This result makes use of `steps-sat-isHighestN` in `continuity3.lagda`, which was used to prove `WCPp` in [11], and in particular to prove that  $F(\text{upd}(\delta, [\mathbf{s}]))$  keeps track in  $\delta$  of the highest number that  $\mathbf{s}$  is applied to in the computation it performs. The modulus of continuity  $k$  of  $F$  at  $\text{upd}(\delta, [\mathbf{s}])$  is then the value stored by  $\delta$  at the end of this computation.

Therefore, because the  $k^{\text{th}}$  iteration of `loop(F) 0 α0` runs  $F(\text{upd}(\delta, [\mathbf{s}]))$  such that  $\delta$ 's value stays under  $k$ , it returns  $\eta(i)$  for some  $i$ , which contradicts the assumption that the branch is infinite, i.e., contains only  $F$ 's (see `noInfPath` in `barContP6.lagda` for details).

Note that the  $k^{\text{th}}$  iteration of `loop(F) 0 α0` does not quite run  $F(\text{upd}(\delta, [\mathbf{s}]))$ , but instead  $F(\text{upd}(\delta, \alpha))$ , where as indicated in Sec. 5.1,  $\alpha$  is built starting from  $\alpha_0$  using the `append` function, and therefore is equal to  $[\mathbf{s}]$  up to  $k$ . We can interchangeably use  $F(\text{upd}(\delta, [\mathbf{s}]))$  or  $F(\text{upd}(\delta, \alpha))$  thanks to Lem. 8 below (see `updSeq-steps-NUM` in `barContP6.lagda`).

► **Definition 7.** *The simulation relation  $t_1 \approx_{\delta, \mathbf{s}, n} t_2$  holds iff*

$$\begin{aligned} & (t_1 = \text{upd}(\delta, \mathbf{s}) \wedge t_2 = \text{upd}(\delta, s2l(\mathbf{s}, n))) \vee (t_1 = \text{upd}(\delta, s2l(\mathbf{s}, n)) \wedge t_2 = \text{upd}(\delta, \mathbf{s})) \\ \vee & (t_1 = x \wedge t_2 = x) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \vee (t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \approx_{\delta, \mathbf{s}, n} b) \\ \vee & (t_1 = (a_1 \ b_1) \wedge t_2 = (a_2 \ b_2) \wedge a_1 \approx_{\delta, \mathbf{s}, n} b_1 \wedge a_2 \approx_{\delta, \mathbf{s}, n} b_2) \vee \dots \end{aligned}$$

where  $s2l(\mathbf{s}, 0) \equiv \alpha_0$  and  $s2l(\mathbf{s}, n+1) \equiv \text{append}(n, s2l(\mathbf{s}, n), \mathbf{s}(n+1))$ .

Most cases are omitted in this definition as they are similar to the ones presented above. Crucially terms of the form  $\delta$  or  $\nu x.t$  are not related, and those are the only expressions not related, thereby ruling out names except when occurring inside `upd` through the first clause.

► **Lemma 8.** *If  $a \approx_{\delta, \mathbf{s}, n} b$  and  $a \xrightarrow{w_1}^*_{w_2} \underline{k}$  such that  $n$  is higher than any value held by  $\delta$  throughout this computation, then  $b \xrightarrow{w_1}^*_{w_2} \underline{k}$ .*

## 5.6 Step 5: The Continuity Property

It now remains to prove that given  $F \in \mathfrak{B} \rightarrow \text{Nat}$ , the tree  $d \equiv (\text{loop}(F) \ 0 \ \alpha_0) \in \mathbf{Bt}$  satisfies the property  $\prod_p \alpha : \mathfrak{B}. \text{follow}(d, \alpha) = F(\alpha) \in \text{Nat}$  (see `semCond` in `barContP9.lagda`). For this we need to prove that `follow(d, α)` computes to the same number that  $F(\alpha)$  computes to, and this for any pure sequence  $\alpha \in \mathfrak{B}$  and tree  $d \equiv \text{loop}(F) \ \underline{k} \ \alpha_k$ , where  $\alpha_k$  agrees with  $\alpha$  up to  $k$  (see `follow-NUM` in `barContP9.lagda`). We prove this by induction on  $d$ . Either  $d$  is an  $\eta(i)$ , which we discuss below, or a  $F(f)$ , in which case we conclude by induction. In case  $d$  is  $\eta(i)$ , we must prove that  $F(\alpha)$  computes to  $i$ . In that case,  $d$  runs  $F(\text{upd}(\delta, \alpha_k))$  for some “fresh”  $\delta$ , which computes to  $i$  for some  $\alpha_k$  that agrees with  $\alpha$  up to  $k$ . Here  $\alpha_k$  is  $s2l(\mathbf{s}, k)$ , for some  $\mathbf{s}$  equal to  $\alpha$  in  $\mathfrak{B}$ . We use again here a metatheoretical sequence  $\mathbf{s}$ , which does not have any computational purpose. We can then prove that  $F(\alpha)$  and  $F(\mathbf{s})$  compute to the same number, and appealing to Lem. 8, we prove that  $F(\mathbf{s})$  and  $F(\text{upd}(\delta, \alpha_k))$  compute to the same number, and therefore that  $F(\alpha)$  computes to  $i$ , which concludes our proof.

## 6 Conclusion and Related Works

The paper presents the first internalization of the inductive dialogue-based continuity principle in a dependent type theory, namely  $\text{TT}_{\mathcal{C}}^{\square}$ , which has been formalized in Agda. For this, we construct Brouwer trees via effectful computations that use references. Proving the inductive continuity principle internally entails new challenges, such as the termination proof which requires maintaining a strict connection between a meta-theoretical generic element and an internal computation. More generally, the class of effectful intuitionistic theories  $\text{TT}_{\mathcal{C}}^{\square}$ ,

which now internalizes several continuity principles, provides a computational framework for further studying the relationship between these principles. WCP and ICP have been shown to coincide in the presence of Bar Induction (under certain restrictions), or assuming classical reasoning [6, 22, 9]. Bar Induction was shown to be consistent with a subsystem of  $\text{TT}_{\mathcal{C}}^{\square}$  [34]. Thus, it seems that  $\text{TT}_{\mathcal{C}}^{\square}$  provides an ideal framework in which one can formally verify this implication internally, as well as produce a corresponding computation. An immediate related question we leave for further study is then to establish the relation between the two principles in a general setting, without assuming Bar Induction or resorting to classical reasoning.

The technique of using dialogue trees to compute moduli of continuity originated in [20, 22, 21, 19], while the idea of recording the interaction of a function with an oracle to compute continuity goes back to Longley [28], where exceptions and references were used as a probing mechanism to compute moduli of continuity. In [19], Escardó defined a model of System T where  $\mathbb{N}$  is interpreted as the type of dialogue trees and function types as functions between the interpretations of the source and target types. This model contains a *generic element* of type  $\mathbb{N} \rightarrow \mathbb{N}$ , a function from dialogue trees to dialogue trees, that records queries to it in the structure of the resulting dialogue tree. Then, a dialogue tree is built using this generic element, from which the modulus of continuity can be calculated. Sterling [35] extended the effectful forcing technique to prove that System T validates the realizable bar thesis, which is equivalent to the inductive continuity principle considered here. System T was given a call-by-name interpretation, where types are interpreted as algebras over a dialogue tree monad. Although the carrier sets of this interpretation agree with those of Escardó, the actions of the algebras allow for a compositional interpretation of the recursor on numbers.

In [3], the authors prove that all BTT [29] functions are continuous by generalizing the method of [19]. However, their method does not allow *internalizing* the continuity principle, which is the goal of the present work. As they work in the metatheory, they can induct on the syntax of the  $F \in \mathfrak{B} \rightarrow \text{Nat}$  when constructing the dialogue trees, allowing for a constructive proof of continuity. In this work, we construct a program computing such trees in the theory itself, where recursion on syntax of terms is not available. As a result we resort to classical logic to prove finiteness of the computed trees and termination of this program. It remains to be seen if this can also be done internally, without resorting to classical logic.

---

## References

- 1 Michael Gordon Abbott. *Categories of containers*. PhD thesis, University of Leicester, England, UK, 2003. URL: <https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.401007>.
- 2 Agda wiki. URL: <http://wiki.portal.chalmers.se/agda/pmwiki.php>.
- 3 Martin Baillon, Assia Mahboubi, and Pierre-Marie Pédro. Gardening with the pythia A model of continuity in a dependent setting. In *CSL*, volume 216 of *LIPICs*, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CSL.2022.5.
- 4 Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.
- 5 Stefano Berardi, Marc Bezem, and Thierry Coquand. On the computational content of the axiom of choice. *J. Symb. Log.*, 63(2):600–622, 1998. doi:10.2307/2586854.
- 6 Nuria Brede and Hugo Herbelin. On the logical structure of choice and bar induction principles. In *LICS*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470523.
- 7 Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987.
- 8 L.E.J. Brouwer. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, chapter On the Domains of Definition of Functions, pages 1923b, 1954, and 1954a. Harvard University Press, 1927.

- 9 Venanzio Capretta and Tarmo Uustalu. A coalgebraic view of bar recursion and bar induction. In Bart Jacobs and Christof Löding, editors, *FOSSACS*, volume 9634 of *LNCS*, pages 91–106. Springer, 2016. doi:10.1007/978-3-662-49630-5\_6.
- 10 Liron Cohen and Vincent Rahli. Constructing unprejudiced extensional type theories with choices via modalities. In *FSCD*, volume 228 of *LIPICs*, pages 10:1–10:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.FSCD.2022.10.
- 11 Liron Cohen and Vincent Rahli. Realizing continuity using stateful computations. In *CSL*, volume 252 of *LIPICs*, pages 15:1–15:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CSL.2023.15.
- 12 Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Inform.*, 100(1-4):43–52, 2010. doi:10.3233/FI-2010-262.
- 13 Thierry Coquand and Guilhem Jaber. A computational interpretation of forcing in type theory. In *Epistemology versus Ontology*, volume 27 of *Logic, Epistemology, and the Unity of Science*, pages 203–213. Springer, 2012. doi:10.1007/978-94-007-4435-6\_10.
- 14 M. J. Cresswell and G. E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.
- 15 Michael A. E. Dummett. *Elements of Intuitionism*. Clarendon Press, second edition, 2000.
- 16 Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *TLCA*, volume 1581 of *LNCS*, pages 129–146. Springer, 1999. doi:10.1007/3-540-48959-2\_11.
- 17 Martín Escardó and Paulo Oliva. Dialogue to brouwer, 2017. URL: <https://www.cs.bham.ac.uk/~mhe/dialogue/dialogue-to-brouwer.html>.
- 18 Martín H. Escardó and Chuangjie Xu. The inconsistency of a Brouwerian continuity principle with the Curry-Howard interpretation. In *TLCA*, volume 38 of *LIPICs*, pages 153–164. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPICs.TLCA.2015.153.
- 19 Martín Hötzel Escardó. Continuity of Gödel’s system T definable functionals via effectful forcing. *Electr. Notes Theor. Comput. Sci.*, 298:119–141, 2013. doi:10.1016/j.entcs.2013.09.010.
- 20 Neil Ghani, Peter G. Hancock, and Dirk Pattinson. Continuous functions on final coalgebras. In *CMCS*, volume 164 of *ENTCS*, pages 141–155. Elsevier, 2006. doi:10.1016/j.entcs.2006.06.009.
- 21 Neil Ghani, Peter G. Hancock, and Dirk Pattinson. Continuous functions on final coalgebras. In Samson Abramsky, Michael W. Mislove, and Catuscia Palamidessi, editors, *MFPS*, volume 249 of *ENTCS*, pages 3–18. Elsevier, 2009. doi:10.1016/j.entcs.2009.07.081.
- 22 Neil Ghani, Peter G. Hancock, and Dirk Pattinson. Representations of stream processors using nested fixed points. *Log. Methods Comput. Sci.*, 5(3), 2009. URL: <http://arxiv.org/abs/0905.4813>.
- 23 S.C. Kleene. Recursive functionals and quantifiers of finite types revisited i. In *Generalized Recursion Theory II*, volume 94 of *Studies in Logic and the Foundations of Mathematics*, pages 185–222. Elsevier, 1978. doi:10.1016/S0049-237X(08)70933-9.
- 24 Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions*. North-Holland Publishing Company, 1965.
- 25 Georg Kreisel. On weak completeness of intuitionistic predicate logic. *J. Symb. Log.*, 27(2):139–158, 1962. doi:10.2307/2964110.
- 26 Saul A. Kripke. Semantical analysis of modal logic i. normal propositional calculi. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963. doi:10.1002/malq.19630090502.
- 27 Saul A. Kripke. Semantical analysis of intuitionistic logic i. In *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. Elsevier, 1965. doi:10.1016/S0049-237X(08)71685-9.
- 28 John Longley. When is a functional program not a functional program? In *ICFP*, pages 1–7. ACM, 1999. doi:10.1145/317636.317775.

- 29 Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. In *LICS*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005113.
- 30 Andrew M Pitts. Nominal sets: Names and symmetry in computer science, volume 57 of *Cambridge tracts in theoretical computer science*, 2013.
- 31 Vincent Rahli. Exercising nuprl’s open-endedness. In *ICMS*, volume 9725 of *LNCS*, pages 18–27. Springer, 2016. doi:10.1007/978-3-319-42432-3\_3.
- 32 Vincent Rahli and Mark Bickford. A nominal exploration of intuitionism. In Jeremy Avigad and Adam Chlipala, editors, *CPP*, pages 130–141. ACM, 2016. doi:10.1145/2854065.2854077.
- 33 Vincent Rahli and Mark Bickford. Validating brouwer’s continuity principle for numbers using named exceptions. *MSCS*, pages 1–49, 2017. doi:10.1017/S0960129517000172.
- 34 Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. Bar induction is compatible with constructive type theory. *J. ACM*, 66(2):13:1–13:35, 2019. doi:10.1145/3305261.
- 35 Jonathan Sterling. Higher order functions and brouwer’s thesis. *J. Funct. Program.*, 31:e11, 2021. doi:10.1017/S0956796821000095.
- 36 Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics An Introduction*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1988.
- 37 A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. New York, Springer, 1973.
- 38 A.S. Troelstra. A note on non-extensional operations in connection with continuity and recursiveness. *Indagationes Mathematicae*, 39(5):455–462, 1977. doi:10.1016/1385-7258(77)90060-9.
- 39 Wim Veldman. Understanding and using Brouwer’s continuity principle. In *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*, pages 285–302. Springer Netherlands, 2001. doi:10.1007/978-94-015-9757-9\_24.
- 40 Chuangjie Xu. *A continuous computational interpretation of type theories*. PhD thesis, University of Birmingham, UK, 2015. URL: <http://etheses.bham.ac.uk/5967/>.
- 41 Chuangjie Xu and Martín Hötzel Escardó. A constructive model of uniform continuity. In *TLCA*, volume 7941 of *LNCS*, pages 236–249. Springer, 2013. doi:10.1007/978-3-642-38946-7\_18.

# Approximating the Value of Energy-Parity Objectives in Simple Stochastic Games

Mohan Dantam

School of Informatics, University of Edinburgh, UK

Richard Mayr

School of Informatics, University of Edinburgh, UK

---

## Abstract

We consider simple stochastic games  $\mathcal{G}$  with energy-parity objectives, a combination of quantitative rewards with a qualitative parity condition. The Maximizer tries to avoid running out of energy while simultaneously satisfying a parity condition.

We present an algorithm to approximate the value of a given configuration in 2-NEXPTIME. Moreover,  $\varepsilon$ -optimal strategies for either player require at most  $\mathcal{O}\left(2\text{-EXP}(|\mathcal{G}|) \cdot \log\left(\frac{1}{\varepsilon}\right)\right)$  memory modes.

**2012 ACM Subject Classification** Computing methodologies → Stochastic games

**Keywords and phrases** Energy-Parity Games, Simple Stochastic Games, Parity, Energy

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.38

**Related Version** *Full Version:* <http://arxiv.org/abs/2307.05762>

## 1 Introduction

**Background.** *Simple stochastic games* (SSGs) are 2-player turn-based perfect information stochastic games played on finite graphs. They are also called *competitive Markov decision processes* [20], or  *$2\frac{1}{2}$ -player games* [13, 12]. Introduced by Shapley [36] in 1953, they have since played a central role in the solution of many problems, e.g., synthesis of reactive systems [35, 34] and formal specification and verification [17, 18, 1]. Every state either belongs to one of the players (Maximizer or Minimizer) or is a random state. In each round of the game the player who owns the current state gets to choose the successor state along the game graph. For random states the successor is chosen according to a predefined distribution. Given a start state and strategies of Maximizer and Minimizer, this yields a distribution over induced infinite plays. We consider objectives  $\mathbf{0}$  that are measurable subsets of the set of possible plays, and the players try to maximize (resp. minimize) the probability of  $\mathbf{0}$ .

Many different objectives for SSGs have been studied in the literature. Here we focus on parity, mean-payoff and energy objectives. We assign numeric rewards to transitions and priorities (aka colors), encoded by bounded non-negative numbers, to states. A play satisfies the (min-even) *parity objective* iff the minimal priority that appears infinitely often in a play is even. It subsumes all  $\omega$ -regular objectives, and in particular safety, liveness, fairness, etc. On finite SSGs, the parity objective can be seen as a special case of the *mean-payoff objective* which requires the limit average reward per transition along a play to be positive (or non-negative). Mean-payoff objectives in SSGs go back to a 1957 paper by Gillette [21] and have been widely studied, due to their relevance for efficient control. The *energy objective* [6] requires that the accumulated reward at any time in a play stays above some finite threshold. The intuition is that a controlled system has some finite initial energy level that must never become depleted. Since the accumulated reward is not bounded a-priori, this essentially turns a finite-state game into an infinite-state one.



© Mohan Dantam and Richard Mayr;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 38; pp. 38:1–38:15

Leibniz International Proceedings in Informatics



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

**Energy-parity.** We consider SSGs with *energy-parity* objectives, where plays need to satisfy both an energy and a parity objective. The parity objective specifies functional correctness, while the energy condition can encode efficiency or risk considerations, e.g., the system should not run out of energy since manually recharging would be costly or risky.

**Previous work.** Much work on combined objectives for stochastic systems is restricted to Markov decision processes (MDPs) [8, 9, 4, 28].

For (stochastic) games, the computational complexity of single objectives is often in  $\text{NP} \cap \text{coNP}$ , e.g., for parity or mean-payoff objectives [25]. Multi-objective games can be harder, e.g., satisfying *two different* parity objectives leads to  $\text{coNP}$  completeness [11].

Stochastic mean-payoff parity games can be solved in  $\text{NP} \cap \text{coNP}$  [10]. However, this does not imply a solution for stochastic energy-parity games, since, unlike in the non-stochastic case [7], there is no known reduction from energy-parity to mean-payoff parity in stochastic games. The reduction in [7] relies on the fact that Maximizer has a winning *finite-memory* strategy for energy-parity, which does not generally hold for stochastic games, or even MDPs [28]. For the same reason, the direct reduction from stochastic energy-parity to ordinary energy games proposed in [8, 9] does not work for general energy-parity but only for energy-Büchi; cf. [28].

Non-stochastic energy-parity games can be solved in  $\text{NP} \cap \text{coNP}$  (and even in pseudo-quasi-polynomial time [16]) and Maximizer strategies require only finite (but exponential) memory [7].

Stochastic energy-parity games have been studied in [29], where it was shown that the almost-sure problem is decidable and in  $\text{NP} \cap \text{coNP}$ . That is, given an initial configuration (control-state plus current energy level), does Maximizer have a strategy to ensure that energy-parity is satisfied with probability 1 against any Minimizer strategy? Unlike in many single-objective games, such an almost-surely winning Maximizer strategy (if it exists) requires infinite memory in general. This holds even in MDPs and for energy-coBüchi objectives [28].

However, [29] did not address quantitative questions about energy-parity objectives, such as computing/approximating the value of a given configuration, or the decidability of exact questions like “Is the value of this configuration  $\geq k$  ?” for some constant  $k$  (e.g.,  $k = 1/2$ ).

The decidability of the latter type of exact question about the energy-parity value is open, but there are strong indications that it is very hard. In fact, even simpler subproblems are already at least as hard as the *positivity problem for linear recurrence sequences*, which in turn is at least as hard as the *Skolem problem* [19]. (The decidability of these problems has been open for decades; see [30] for an overview.) Given an SSG with an energy-parity objective, suppose we remove the parity condition (assume it is always true) and also suppose that Maximizer is passive (does not get to make any decisions). Then we obtain an MDP where the only active player (the Minimizer in the SSG) has a *termination objective*, i.e., to reach a configuration where the energy level is  $\leq 0$ . Exact questions about the value of the termination objective in MDPs are already at least as hard as the positivity problem [31, Section 5.2.3] (see also [32, 33]). Thus exact questions about the energy-parity value in SSGs are also at least as hard as the positivity problem.

**Our contributions.** Since exact questions about the energy-parity value in SSGs are positivity-hard, we consider the problem of computing approximations of the value. We present an algorithm that, given an SSG  $\mathcal{G}$  and error  $\varepsilon$ , computes  $\varepsilon$ -close approximations of the energy-parity value of any given configuration in 2-NEXPTIME. Moreover, we show that

$\varepsilon$ -optimal Maximizer (resp. Minimizer) strategies can be chosen as deterministic and using only finite memory with  $\mathcal{O}(2\text{-EXP}(|\mathcal{G}|) \cdot \log(\frac{1}{\varepsilon}))$  memory modes. One can understand the idea as a constructive upper bound on the accuracy with which the players need to remember the current energy level in the game. (This is in contrast to the result in [28] that almost-surely winning Maximizer strategies require infinite memory in general.) Once the upper bound on Maximizer's memory for  $\varepsilon$ -optimal strategies is established, one might attempt a reduction from energy-parity to mean-payoff parity, along similar lines as for non-stochastic games in [7]. However, instead we use a more direct reduction from energy-parity to parity in a derived SSG for our approximation algorithm.

## 2 Preliminaries

A *probability distribution* over a countable set  $S$  is a function  $f: S \rightarrow [0, 1]$  with  $\sum_{s \in S} f(s) = 1$ .  $\text{supp}(f) \stackrel{\text{def}}{=} \{s \mid f(s) > 0\}$  denotes the support of  $f$  and  $\mathcal{D}(S)$  is the set of all probability distributions over  $S$ . Given an alphabet  $\Sigma$ , let  $\Sigma^\omega$  and  $\Sigma^*$  ( $\Sigma^+$ ) denote the set of infinite and finite (non-empty) sequences over  $\Sigma$ , respectively. Elements of  $\Sigma^\omega$  or  $\Sigma^*$  are called words.

**Games, MDPs and Markov chains.** A *Simple Stochastic Game* (SSG) is a finite-state 2-player turn-based perfect-information stochastic game  $\mathcal{G} = (S, (S_\square, S_\diamond, S_\circ), E, P)$  where the finite set of states  $S$  is partitioned into the states  $S_\square$  of the player  $\square$  (*Maximizer*), states  $S_\diamond$  of player  $\diamond$  (*Minimizer*), and chance vertices (aka random states)  $S_\circ$ . Let  $E \subseteq S \times S$  be the transition relation. We write  $s \rightarrow s'$  if  $(s, s') \in E$  and assume that  $\text{Succ}(s) \stackrel{\text{def}}{=} \{s' \mid sEs'\} \neq \emptyset$  for every state  $s$ . The *probability function*  $P$  assigns each random state  $s \in S_\circ$  a distribution over its successor states, i.e.,  $P(s) \in \mathcal{D}(\text{Succ}(s))$ . For ease of presentation, we extend the domain of  $P$  to  $S^*S_\circ$  by  $P(\rho s) \stackrel{\text{def}}{=} P(s)$  for all  $\rho s \in S^*S_\circ$ . An *MDP* is a game where one of the two players does not control any states. An MDP is *maximizing* (resp. *minimizing*) iff  $S_\diamond = \emptyset$  (resp.  $S_\square = \emptyset$ ). A *Markov chain* is a game with only random states, i.e.,  $S_\square = S_\diamond = \emptyset$ .

**Strategies.** A *play* is an infinite sequence  $s_0s_1 \dots \in S^\omega$  such that  $s_i \rightarrow s_{i+1}$  for all  $i \geq 0$ . A *path* is a finite prefix of a play. Let  $\text{Plays}(\mathcal{G}) \stackrel{\text{def}}{=} \{\rho = (q_i)_{i \in \mathbb{N}} \mid q_i \rightarrow q_{i+1}\}$  denote the set of all possible plays. A strategy of the player  $\square$  ( $\diamond$ ) is a function  $\sigma: S^*S_\square \rightarrow \mathcal{D}(S)$  ( $\pi: S^*S_\diamond \rightarrow \mathcal{D}(S)$ ) that assigns to every path  $ws \in S^*S_\square$  ( $\in S^*S_\diamond$ ) a probability distribution over the successors of  $s$ . If these distributions are always Dirac then the strategy is called *deterministic* (aka pure), otherwise it is called *randomized* (aka mixed). The set of all strategies of player  $\square$  and  $\diamond$  in  $\mathcal{G}$  is denoted by  $\Sigma_{\mathcal{G}}$  and  $\Pi_{\mathcal{G}}$ , respectively. A play/path  $s_0s_1 \dots$  is compatible with a pair of strategies  $(\sigma, \pi)$  if  $s_{i+1} \in \text{supp}(\sigma(s_0 \dots s_i))$  whenever  $s_i \in S_\square$  and  $s_{i+1} \in \text{supp}(\pi(s_0 \dots s_i))$  whenever  $s_i \in S_\diamond$ .

Finite-memory deterministic (FD) strategies are a subclass of strategies described by deterministic transducers  $T = (M, m_0, \text{upd}, \text{nxt})$  where  $M$  is a finite set of memory modes with initial mode  $m_0$ ,  $\text{upd}: M \times E \mapsto M$  updates the memory mode upon observing a transition and  $\text{nxt}: M \times S_\circ \mapsto S$  chooses the successor state based on the current memory mode and state. FD strategies without memory ( $|M| = 1$ ) are called memoryless deterministic (MD). For deterministic strategies, there is no difference between public memory (observable by the other player) and private memory.

**Measure.** A game  $\mathcal{G}$  with initial state  $s_0$  and strategies  $(\sigma, \pi)$  yields a probability space  $(s_0S^\omega, \mathcal{F}_{s_0}, \mathcal{P}_{\sigma, \pi, s_0}^{\mathcal{G}})$  where  $\mathcal{F}_{s_0}$  is the  $\sigma$ -algebra generated by the cylinder sets  $s_0s_1 \dots s_nS^\omega$  for  $n \geq 0$ . The probability measure  $\mathcal{P}_{\sigma, \pi, s_0}^{\mathcal{G}}$  is first defined on the cylinder sets. For

$\rho = s_0 \dots s_n$ , let  $\mathcal{P}_{\sigma, \pi, s_0}^{\mathcal{G}}(\rho) \stackrel{\text{def}}{=} 0$  if  $\rho$  is not compatible with  $\sigma, \pi$  and otherwise  $\mathcal{P}_{\sigma, \pi, s_0}^{\mathcal{G}}(\rho S^\omega) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} \tau(s_0 \dots s_i)(s_{i+1})$  where  $\tau$  is  $\sigma$  or  $\pi$  or  $P$  depending on whether  $s_i \in S_\square$  or  $S_\diamond$  or  $S_\circ$ , respectively. By Carathéodory's extension theorem [2], this defines a unique probability measure on the  $\sigma$ -algebra.

**Objectives and Payoff functions.** General objectives are defined by real-valued measurable functions. However, we only consider indicator functions of measurable sets. Hence our objectives can be described by measurable subsets  $\mathbf{0} \subseteq S^\omega$  of plays. The payoff, under strategies  $(\sigma, \pi)$ , is the probability that plays belong to  $\mathbf{0}$ .

We use the syntax and semantics of the LTL operators [14] **F** (eventually), **G** (always) and **X** (next) to specify some conditions on plays.

*Reachability & Safety.* A reachability objective is defined by a set of target states  $T \subseteq S$ . A play  $\rho = s_0 s_1 \dots$  belongs to **FT** iff  $\exists i \in \mathbb{N} s_i \in T$ . Similarly,  $\rho$  belongs to  $\mathbf{F}^{\leq n} T$  (resp.  $\mathbf{F}^{\geq n} T$ ) iff  $\exists i \leq n$  (resp.  $i \geq n$ ) such that  $s_i \in T$ . Dually, the safety objective **GT** consists of all plays which never leave  $T$ . We have  $\mathbf{GT} = \neg \mathbf{F}\neg T$ .

*Parity.* A parity objective is defined via bounded function  $Col : S \rightarrow \mathbb{N}$  that assigns non-negative priorities (aka colors) to states. Given an infinite play  $\rho = s_0 s_1 \dots$ , let  $\text{Inf}(\rho)$  denote the set of numbers that occur infinitely often in the sequence  $Col(s_0) Col(s_1) \dots$ . A play  $\rho$  satisfies *even parity* w.r.t.  $Col$  iff the minimum of  $\text{Inf}(\rho)$  is even. Otherwise,  $\rho$  satisfies *odd parity*. The objective even parity is denoted by **EPAR**( $Col$ ) and odd parity is denoted by **OPAR**( $Col$ ). Most of the time, we implicitly assume that the coloring function is known and just write **EPAR** and **OPAR**. Observe that, given any coloring  $Col$ , we have  $\overline{\mathbf{EPAR}} = \mathbf{OPAR}$  and  $\mathbf{OPAR}(Col) = \mathbf{EPAR}(Col + 1)$  where  $Col + 1$  is the function which adds 1 to the color of every state. This justifies to consider only one of the even/odd parity objectives, but, for the sake of clarity, we distinguish these objectives wherever necessary.

*Energy/Reward/Counter based objectives.* Let  $r : E \rightarrow \{-R, \dots, 0, \dots, R\}$  be a bounded function that assigns weights to transitions. Depending on context, the sum of these weights in a path can be viewed as energy, cost/reward or a counter. If  $s \xrightarrow{c} s'$  and  $r((s, s')) = c$ , we write  $s \xrightarrow{c} s'$ . Let  $\rho = s_0 \xrightarrow{c_0} s_1 \xrightarrow{c_1} \dots$  be a play. We say that  $\rho$  satisfies

1. the  $k$ -energy objective **EN**( $k$ ) iff  $\left(k + \sum_{i=0}^{n-1} c_i\right) > 0$  for all  $n \geq 0$ .
2. the  $l$ -storage condition if  $l + \sum_{i=m}^{n-1} c_i \geq 0$  holds for every infix  $s_m \xrightarrow{c_m} s_{m+1} \dots s_n$  of the play. Let **ST**( $k, l$ ) denote the set of plays that satisfy both the  $k$ -energy and the  $l$ -storage condition. Let  $\mathbf{ST}(k) \stackrel{\text{def}}{=} \bigcup_l \mathbf{ST}(k, l)$ . Clearly,  $\mathbf{ST}(k) \subseteq \mathbf{EN}(k)$ .
3.  $k$ -Termination **Term**( $k$ ) iff there exists  $n \geq 0$  such that  $\left(k + \sum_{i=0}^{n-1} c_i\right) \leq 0$ .
4. Limit objective **LimInf**( $\triangleright z$ ) iff  $\left(\liminf_{n \rightarrow \infty} \sum_{i=0}^{n-1} c_i\right) \triangleright z$  for  $\triangleright \in \{<, \leq, =, \geq, >\}$  and  $z \in \mathbb{R} \cup \{\infty, -\infty\}$  and similarly for **LimSup**( $\triangleright z$ ).
5. Mean payoff **MP**( $\triangleright c$ ) for some constant  $c \in \mathbb{R}$  iff  $\left(\liminf_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} c_i\right) \triangleright c$ .

Observe that the objectives  $k$ -energy and  $k$ -termination are mutually exclusive and cover all of the plays. A different way to consider these objectives is to encode the energy level (the sum of the transition weights so far) into the state space and then consider the obtained infinite-state game with safety/reachability objective, respectively.

An objective  $\mathbf{0}$  is called *shift-invariant* iff for all finite paths  $\rho$  and plays  $\rho' \in S^\omega$ , we have  $\rho\rho' \in \mathbf{0} \iff \rho' \in \mathbf{0}$ . Parity and mean payoff objectives are shift-invariant, but energy and termination objectives are not. Objective  $\mathbf{0}$  is called *submixing* iff for all sequences of finite non-empty words  $u_0, v_0, u_1, v_1 \dots$  we have  $u_0 v_0 u_1 v_1 \dots \in \mathbf{0} \implies ((u_0 u_1 \dots \in \mathbf{0}) \vee (v_0 v_1 \dots \in \mathbf{0}))$ .

**Determinacy.** Given an objective  $\mathbf{0}$  and a game  $\mathcal{G}$ , state  $s$  has value (w.r.t to  $\mathbf{0}$ ) iff

$$\sup_{\sigma \in \Sigma_{\mathcal{G}}} \inf_{\pi \in \Pi_{\mathcal{G}}} \mathcal{P}_{\sigma, \pi, s}^{\mathcal{G}}(\mathbf{0}) = \inf_{\pi \in \Pi_{\mathcal{G}}} \sup_{\sigma \in \Sigma_{\mathcal{G}}} \mathcal{P}_{\sigma, \pi, s}^{\mathcal{G}}(\mathbf{0}).$$

If  $s$  has value then  $\text{val}_{\mathbf{0}}^{\mathcal{G}}(s)$  denotes the value of  $s$  defined by the above equality. A game with an objective is called *weakly determined* if every state has value. Stochastic games with Borel objectives are weakly determined [26, 27]. Our objectives above are Borel, hence any boolean combination of them is also weakly determined. For  $\varepsilon > 0$  and state  $s$ , a strategy

1.  $\sigma \in \Sigma_{\mathcal{G}}$  is  $\varepsilon$ -optimal (maximizing) iff  $\mathcal{P}_{\sigma, \pi, s}^{\mathcal{G}}(\mathbf{0}) \geq \text{val}_{\mathbf{0}}^{\mathcal{G}}(s) - \varepsilon$  for all  $\pi \in \Pi_{\mathcal{G}}$ .
2.  $\pi \in \Pi_{\mathcal{G}}$  is  $\varepsilon$ -optimal (minimizing) iff  $\mathcal{P}_{\sigma, \pi, s}^{\mathcal{G}}(\mathbf{0}) \leq \text{val}_{\mathbf{0}}^{\mathcal{G}}(s) + \varepsilon$  for all  $\sigma \in \Sigma_{\mathcal{G}}$ .

A 0-optimal strategy is called *optimal*. An MD strategy is called *uniformly  $\varepsilon$ -optimal* (resp. uniformly optimal) if it is so from every start state. An optimal strategy for player  $\square$  from state  $s$  is *almost surely winning* if  $\text{val}_{\mathbf{0}}^{\mathcal{G}}(s) = 1$ . By AS ( $\mathbf{0}$ ) we denote the set of states that have an almost surely winning strategy for objective  $\mathbf{0}$ . For ease of presentation, we drop subscripts and superscripts wherever possible if they are clear from the context.

**Energy-parity.** We are concerned with approximating the value for the combined energy-parity objective  $\text{EN}(k) \cap \text{EPAR}$  and building  $\varepsilon$ -optimal strategies.

In our constructions we use some auxiliary objectives. Following [29], these are defined as  $\text{Gain} \stackrel{\text{def}}{=} \text{LimInf}(\triangleright -\infty) \cap \text{EPAR}$  and  $\text{Loss} \stackrel{\text{def}}{=} \overline{\text{Gain}} = \text{LimInf}(=\infty) \cup \text{OPAR}$ .

► **Remark 1.** For finite-state SSGs and the following objectives there exist optimal MD strategies for both players. Moreover, if the SSG is just a maximizing MDP then the set of states that are almost surely winning for Maximizer can be computed in polynomial time.

1. FT [15]
2.  $\text{LimInf}(\triangleright -\infty)$ ,  $\text{LimInf}(\triangleright \infty)$ ,  $\text{LimSup}(\triangleright -\infty)$ ,  $\text{LimSup}(\triangleright \infty)$ ,  $\text{MP}(\triangleright 0)$  [5, Prop. 1]
3. EPAR [37]

### 3 The Main Result

The following theorem states our main result.

► **Theorem 2.** Let  $\mathcal{G} = (S, (S_{\square}, S_{\diamond}, S_{\circ}), E, P)$  be an SSG with transition rewards in unary assigned by function  $r$  and colors assigned to states by function  $\text{Col}$ . For every state  $s \in S$ , initial energy level  $i \geq 0$  and error margin  $\varepsilon > 0$ , one can compute

1. a rational number  $v'$  such that  $0 \leq v' - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon$  in 2-NEXPTIME.<sup>1</sup>
2.  $\varepsilon$ -optimal FD strategies  $\sigma_{\varepsilon}$  and  $\pi_{\varepsilon}$  for Maximizer and Minimizer, resp., in 2-NEXPTIME.

These strategies use  $\mathcal{O}(2\text{-EXP}(|\mathcal{G}|) \cdot \log(\frac{1}{\varepsilon}))$  memory modes.

For rewards in binary, the bounds above increase by one exponential.

We outline the main steps of the proof; details in the following sections. We begin with the observation that  $\text{EN}(i) \subseteq \text{EN}(j)$  for  $i \leq j$ , and thus for all states  $s$  we have  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \text{val}_{\text{EN}(j) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq 1$ . So  $\lim_{n \rightarrow \infty} \text{val}_{\text{EN}(n) \cap \text{EPAR}}^{\mathcal{G}}(s)$  exists. We define

$$\text{Lval}^{\mathcal{G}}(s) \stackrel{\text{def}}{=} \lim_{n \rightarrow \infty} \text{val}_{\text{EN}(n) \cap \text{EPAR}}^{\mathcal{G}}(s). \quad (1)$$

<sup>1</sup> We write “computing a number  $v'$  in 2-NEXPTIME” as a shorthand for the property that questions like  $v' \leq c$  for constants  $c$  are decidable in 2-NEXPTIME.

We will see that  $\text{Lval}^{\mathcal{G}}(s)$  and  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  are in fact equal (a consequence of Lemma 9) and  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  can be computed in nondeterministic polynomial time (Theorem 5). Intuitively, for high energy levels, the precise energy level does not matter much for the value.

The main steps of the approximation algorithm are as follows.

1. Compute FD strategies  $\sigma^*(s)$  that are optimal maximizing for the objective **Gain** starting from state  $s$  in  $\mathcal{G}$ . Compute an MD strategy  $\pi^*$  that is uniformly optimal minimizing for the objective **Gain**. Compute the value  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  for every  $s \in S$ . See Section 4.
2. Compute a natural number  $N$  such that for all  $s \in S$  and all  $i \geq N$  we have

$$0 \leq \text{val}_{\text{Gain}}^{\mathcal{G}}(s) - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon.$$

$N$  will be doubly exponential. See Section 5.

3. Consider the finite-state parity game  $\mathcal{G}'$  derived from  $\mathcal{G}$  by encoding the energy level up-to  $N$  into the states, i.e., the states of  $\mathcal{G}'$  are of the form  $(s, k)$  for  $s \in S$  and  $0 \leq k \leq N$ , and colors are inherited from  $s$ . Moreover, we add gadgets that ensure that states  $(s, N)$  at the upper end win with probability  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  and states  $(s, 0)$  at the lower end lose. By the previous item,  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  is  $\varepsilon$ -close to  $\text{val}_{\text{EN}(N) \cap \text{EPAR}}^{\mathcal{G}}(s)$ . Thus, for  $k < N$  we can  $\varepsilon$ -approximate the value  $v = \text{val}_{\text{EN}(k) \cap \text{EPAR}}^{\mathcal{G}}(s)$  by  $v' \stackrel{\text{def}}{=} \text{val}_{\text{EPAR}}^{\mathcal{G}'}((s, k))$ . If  $k \geq N$  we can  $\varepsilon$ -approximate  $v$  by  $v' \stackrel{\text{def}}{=} \text{val}_{\text{Gain}}^{\mathcal{G}}(s)$ .

Moreover, we obtain  $\varepsilon$ -optimal FD strategies  $\sigma_\varepsilon$  for Maximizer (resp.  $\pi_\varepsilon$  for Minimizer) for  $\text{EN}(k) \cap \text{EPAR}$  in  $\mathcal{G}$ . Let  $\hat{\sigma}$  (resp.  $\hat{\pi}$ ) be optimal MD strategies for Maximizer (resp. Minimizer) for the objective **EPAR** in  $\mathcal{G}'$ . Then  $\sigma_\varepsilon$  plays as follows. While the current energy level  $j$  ( $k$  plus the sum of the rewards so far) stays  $< N$ , then, at any state  $s'$ , play like  $\hat{\sigma}$  at state  $(s', j)$  in  $\mathcal{G}'$ . Once the energy level reaches a value  $\geq N$  at some state  $s'$  for the first time, then play like  $\sigma^*(s')$  forever. Similarly,  $\pi_\varepsilon$  plays as follows. While the current energy level  $j$  ( $k$  plus the sum of the rewards so far) stays  $< N$ , then, at any state  $s'$ , play like  $\hat{\pi}$  at state  $(s', j)$  in  $\mathcal{G}'$ . Once the energy level reaches a value  $\geq N$  (at any state) for the first time, then play like  $\pi^*$  forever. See Section 6.

As a technical tool, we sometimes consider the dual of a game  $\mathcal{G}$  (resp. the dual maximizing MDP of some minimizing MDP). Consider  $\mathcal{G}^d \stackrel{\text{def}}{=} (S', (S'_\square, S'_\diamond, S'_\circ), E', P')$  with the complement objective  $\overline{\text{EN}(k) \cap \text{EPAR}} = \text{Term}(k) \cup \text{OPAR}$ , where  $\mathcal{G}^d$  is simply the game with the roles of Maximizer and Minimizer reversed, i.e.,

$$S' = S \quad S'_\square = S_\diamond \quad S'_\diamond = S_\square \quad S'_\circ = S_\circ \quad E' = E \quad P' = P$$

Hence  $\Sigma_{\mathcal{G}^d} = \Pi_{\mathcal{G}}$  and  $\Pi_{\mathcal{G}^d} = \Sigma_{\mathcal{G}}$ . It is easy to see that for any objective  $\mathbb{O}$  and start state  $s$

1.  $\text{val}_{\mathbb{O}}^{\mathcal{G}}(s) + \text{val}_{\overline{\mathbb{O}}}^{\mathcal{G}^d}(s) = 1$ .
2.  $\sigma$  is  $\varepsilon$ -optimal maximizing for  $\mathbb{O}$  in  $\mathcal{G}$  iff it is  $\varepsilon$ -optimal minimizing for  $\overline{\mathbb{O}}$  in  $\mathcal{G}^d$ .
3.  $\pi$  is  $\varepsilon$ -optimal minimizing for  $\mathbb{O}$  in  $\mathcal{G}$  iff it is  $\varepsilon$ -optimal maximizing for  $\overline{\mathbb{O}}$  in  $\mathcal{G}^d$ .

So approximating the value of  $\text{EN}(k) \cap \text{EPAR}$  in  $\mathcal{G}$  can be reduced in linear time to approximating the value of  $\text{Term}(k) \cup \text{OPAR}$  in  $\mathcal{G}^d$ .

#### 4 Computing $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$

Given an SSG  $\mathcal{G} = (S, (S_\square, S_\diamond, S_\circ), E, P)$  and a start state  $s$ , we will show how to compute  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  and the optimal strategies for both players.

We start with the case of maximizing MDPs. The following lemma summarizes some previous results ([29, Lemmas 30,16], [28, Lemma 26], [24, Proposition 4]).

► **Lemma 3.** *Let  $\mathcal{M}$  be a maximizing MDP.*

1.  $\text{Lval}^{\mathcal{M}}(s) = \text{val}_{\text{Gain}}^{\mathcal{M}}(s)$  for all states  $s \in S$ .
2. *Optimal strategies for Gain in  $\mathcal{M}$  exist and can be chosen FD, with  $\mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}))$  memory modes, and exponential memory is also necessary.*
3. *For any state  $s \in S$ ,  $\text{Lval}^{\mathcal{M}}(s)$  is rational and can be computed in  $\mathcal{O}(|\mathcal{M}|^8)$  deterministic polynomial time if rewards are in unary, and in NP and coNP if rewards are in binary.*

**Proof.** Item 1 holds by [29, Lemma 30].

Towards Item 2, we follow the proof of [29, Lemma 16]. Since  $\text{Gain} = \text{LimInf}( > -\infty ) \cap \text{EPAR}$  is shift-invariant, there exist optimal strategies by [22]. By [28, Theorem 18] and Item 1, an optimal strategy for  $\text{Gain}$  can be constructed as follows. Let  $A \stackrel{\text{def}}{=} \bigcup_{k \in \mathbb{N}} \text{AS}(\text{ST}(k) \cap \text{EPAR})$  and  $B \stackrel{\text{def}}{=} \text{AS}(\text{LimInf}( = \infty ) \cap \text{EPAR})$  be the subsets of states from which there exist almost surely winning strategies for the objectives  $\text{ST}(k) \cap \text{EPAR}$  and  $\text{LimInf}( = \infty ) \cap \text{EPAR}$ , respectively. By [28, Theorem 8], we can restrict the values  $k$  in the definition of  $A$  by some  $k' = \mathcal{O}(|S| \cdot R)$ , i.e.,  $A = \bigcup_{k \leq k'} \text{AS}(\text{ST}(k) \cap \text{EPAR})$ . An optimal strategy  $\sigma$  for  $\text{Gain}$  works in two phases. First it plays an optimal strategy  $\sigma_R$  towards reaching the set  $A \cup B$ , where  $\sigma_R$  can be chosen MD by Remark 1. Then, upon reaching  $A$  (resp.  $B$ ), it plays an almost surely winning strategy  $\sigma_A$  for the objective  $\text{ST}(k) \cap \text{EPAR}$  (resp.  $\sigma_B$  for the objective  $\text{LimInf}( = \infty ) \cap \text{EPAR}$ ). By [28, Theorem 8], the strategy  $\sigma_A$  requires  $\mathcal{O}(k \cdot |S|)$  memory modes for a given  $k$  and thus at most  $\mathcal{O}(|S|^2 \cdot R)$ , since we can assume that  $k \leq k'$ . Towards the strategy  $\sigma_B$ , we first observe that in finite MDPs a strategy is almost-surely winning for  $\text{LimInf}( = \infty ) \cap \text{EPAR}$  iff it is almost-surely winning for  $\text{MP}( > 0 ) \cap \text{EPAR}$ . By [24, Proposition 4], there exist optimal deterministic strategies for  $\text{MP}( > 0 ) \cap \text{EPAR}$  that use exponential memory, i.e.,  $\mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}))$  memory modes. The memory required for  $\sigma_B$  exceeds that of  $\sigma_R$  and  $\sigma_A$  (even when  $R$  is given in binary), and the one extra memory mode to record the switch from  $\sigma_R$  to  $\sigma_A$  (resp.  $\sigma_B$ ) is negligible in comparison. Thus we can conclude that  $\sigma$  uses  $\mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}))$  memory modes. [24, Fig. 1 and Prop. 4] shows that exponential memory is necessary.

Towards Item 3, let  $d \stackrel{\text{def}}{=} |\text{Col}(S)|$  be the number of priorities in the parity condition. By [28, Lemma 26], for each  $s \in S$ ,  $\text{Lval}^{\mathcal{M}}(s)$  is rational and can be computed in deterministic time  $\tilde{\mathcal{O}}(|E| \cdot d \cdot |S|^4 \cdot R + d \cdot |S|^{3.5} \cdot (|P| + |r|)^2)$  (and still in NP and coNP if  $R$  is given in binary). So  $\text{Lval}^{\mathcal{M}}(s)$  can be computed in  $\mathcal{O}(|\mathcal{M}|^8)$  deterministic polynomial time if weights are given in unary, and in NP and coNP if weights are given in binary. ◀

In order to extend Lemma 3 from MDPs to games, we need the notion of derived MDPs, obtained by fixing the choices of one player according to some FD strategy. Given an SSG  $\mathcal{G} = (S, (S_{\square}, S_{\diamond}, S_{\circ}), E, P)$  and a finite memory deterministic (FD) strategy  $\pi$  for Minimizer (resp.  $\sigma$  for Maximizer) from a state  $s$ , described by  $(M, m_0, \text{upd}, \text{nxt})$ , let  $\mathcal{G}_{\pi}$  (resp.  $\mathcal{G}^{\sigma}$ ) be the maximizing (resp. minimizing) MDP with state space  $M \times S$  obtained by fixing Minimizer's (resp. Maximizer's) choices according to  $\pi$  (resp.  $\sigma$ ).

► **Lemma 4.** *For every SSG  $\mathcal{G}$ , objective  $\mathcal{O}$  and Minimizer (resp. Maximizer) FD strategy  $\pi = (M, m_0, \text{upd}, \text{nxt})$  (resp.  $\sigma$ ), from state  $s$  we get  $\text{val}_{\mathcal{O}}^{\mathcal{G}^{\sigma}}((m_0, s)) \leq \text{val}_{\mathcal{O}}^{\mathcal{G}}(s) \leq \text{val}_{\mathcal{O}}^{\mathcal{G}_{\pi}}((m_0, s))$  and equality holds if  $\pi$  (resp.  $\sigma$ ) is optimal from state  $s$ .*

► **Theorem 5.** *Consider an SSG  $\mathcal{G} = (S, (S_{\square}, S_{\diamond}, S_{\circ}), E, P)$  with the Gain objective.*

1. *Optimal Minimizer strategies exist and can be chosen uniform MD.*
2.  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  is rational and questions about it, i.e.,  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s) \leq c$  for constants  $c$ , are decidable in NP.
3. *Optimal Maximizer strategies exist and can be chosen FD, with  $\mathcal{O}(\exp(|\mathcal{G}|^{\mathcal{O}(1)}))$  memory modes. Moreover, exponential memory is also necessary.*

**Proof.** Towards Item 1, observe that since both the objectives  $\text{LimInf}(= -\infty)$  and  $\text{OPAR}$  are shift-invariant and submixing, so is their union, i.e.,  $\overline{\text{Gain}}$  is shift-invariant and submixing. Hence, by [23, Theorem 1.1], an optimal MD strategy  $\pi_s^*$  for Minimizer exists from any state  $s \in S$ . Since  $S$  is finite and  $\text{Gain}$  is shift-invariant, we can also obtain a uniformly optimal MD strategy  $\pi^*$ , i.e.,  $\pi^*$  is optimal from every state.

Towards Item 2, consider the maximizing MDP  $\mathcal{G}_{\pi^*}$  obtained from  $\mathcal{G}$  by fixing  $\pi^*$ . Since  $\pi^*$  is MD, the states of  $\mathcal{G}_{\pi^*}$  are the same as the states as  $\mathcal{G}$ . Since  $\pi^*$  is optimal for Minimizer from every state  $s$ , we obtain that  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s) = \text{val}_{\text{Gain}}^{\mathcal{G}_{\pi^*}}(s)$  for every state  $s$  by Lemma 4. By Lemma 3, the latter is rational and can be computed in polynomial time for weights in unary (resp. in NP and coNP for weights in binary). Thus, by guessing  $\pi^*$ , we can decide questions  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s) \leq c$  in NP.

Towards Item 3, we again use the property that  $\overline{\text{Gain}}$  is shift-invariant and submixing (see above). By [29, Theorem 6, Def. 24], optimal FD Maximizer strategies for  $\text{Gain}$  in an SSG require only  $|S_{\diamond}| \cdot \lceil \log(|E|) \rceil$  many extra bits of memory above the memory required for optimal Maximizer strategies in any derived MDP  $\mathcal{M}$  where Minimizer's choices are fixed. Hence, by Lemma 3, one can obtain optimal FD Maximizer strategies in  $\mathcal{G}$  that use at most  $2^{|S_{\diamond}| \cdot \lceil \log(|E|) \rceil} \cdot \mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)})) = \mathcal{O}(\exp(|\mathcal{G}|^{\mathcal{O}(1)}))$  memory modes. The corresponding exponential lower bound on the memory holds already for MDPs by Lemma 3.  $\blacktriangleleft$

## 5 Computing the Upper Bound $N$

We show how to compute the upper bound  $N$ , up-to which Maximizer needs to remember the energy level, for any given error margin  $\varepsilon > 0$ . Similarly as in Section 4, we first solve the problem for maximizing MDPs and then extend the solution to SSGs.

### 5.1 Computing $N$ for maximizing MDPs

Given a maximizing MDP  $\mathcal{M} = (S, S_{\square}, S_{\circ}, E, P)$  and  $\varepsilon > 0$ , we will compute an  $N \in \mathbb{N}$  such that for all  $s \in S$  and all  $j \geq N$

$$0 \leq \text{val}_{\text{Term}(j) \cup \text{OPAR}}^{\mathcal{M}}(s) - \text{val}_{\text{Loss}}^{\mathcal{M}}(s) \leq \varepsilon.$$

Recall that  $\text{Loss} = \text{LimInf}(= -\infty) \cup \text{OPAR}$ . We now define the sets of states  $W_0 \stackrel{\text{def}}{=} \text{AS}(\text{Loss})$ ,  $W_1 \stackrel{\text{def}}{=} \text{AS}(\text{LimInf}(= -\infty))$  and  $W_2 \stackrel{\text{def}}{=} \text{AS}(\text{OPAR})$ . By Remark 1, there exist optimal MD strategies for  $\text{LimInf}(= -\infty)$  and  $\text{OPAR}$ . Since  $\text{Loss}$  is shift-invariant and submixing, there exists an optimal MD strategy for it by [23, Theorem 1.1].

► **Lemma 6.** *For every state  $s$  in the MDP  $\mathcal{M}$  we have*

1.  $W_1 \cup W_2 \subseteq W_0$
2.  $\text{val}_{\text{FW}_0}(s) \leq \text{val}_{\text{Loss}}(s)$
3.  $\text{val}_{\text{OPAR} \cap \overline{\text{FW}_2}}(s) = 0$
4. *for every initial energy level  $j \geq 0$*

$$\text{val}_{(\text{Term}(j) \cup \text{OPAR}) \cap \text{FW}_0}(s) = \text{val}_{\text{FW}_0}(s) \tag{2}$$

$$\text{val}_{\text{Loss}}(s) \leq \text{val}_{\text{Term}(j) \cup \text{OPAR}}(s) \leq \text{val}_{\text{Loss}}(s) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{\text{FW}_1}) \tag{3}$$

**Proof.**

1. This follows directly from the definitions of  $W_0, W_1, W_2$ .
2. Let  $\sigma'$  be an optimal MD strategy for  $\text{FW}_0$  from  $s$  and  $\sigma''$  be an almost surely winning MD strategy for  $\text{Loss}$  from any state in  $W_0$ . Let  $\sigma$  be the strategy that plays  $\sigma'$  until reaching  $W_0$  and then switches to  $\sigma''$ . We have  $\text{val}_{\text{Loss}}(s) \geq \mathcal{P}_{\sigma, s}(\text{Loss}) \geq \mathcal{P}_{\sigma', s}(\text{FW}_0) = \text{val}_{\text{FW}_0}(s)$ .

3. For  $s \in W_2$  the statement is obvious. So let  $s \notin W_2$  and consider the modified MDP  $\mathcal{M}' = (S', S'_\square, S'_\circ, E', P')$  where all states in  $W_2$  are collapsed into a losing sink. I.e.,  $S' \stackrel{\text{def}}{=} (S \setminus W_2) \uplus \{\text{trap}\}$ , with  $\text{trap}$  a new random sink state having color 0 (thus losing for objective  $\text{OPAR}$ ),  $E'$  contains all of  $(E \cap \{(S \setminus W_2) \times (S \setminus W_2)\} \cup (\text{trap}, \text{trap}))$  and all transitions to  $W_2$  are redirected to  $\text{trap}$  and  $P'$  is derived accordingly from  $P$ . Then  $\text{val}_{\text{OPAR}}^{\mathcal{M}'}(\hat{s}) = \text{val}_{\text{OPAR} \cap \overline{FW_2}}^{\mathcal{M}}(\hat{s})$  for all states  $\hat{s} \in S \setminus W_2$ . Towards a contradiction, assume that  $\text{val}_{\text{OPAR} \cap \overline{FW_2}}^{\mathcal{M}}(s) > 0$ . Hence  $\text{val}_{\text{OPAR}}^{\mathcal{M}'}(s) > 0$ . Then, by [22, Theorem 3.2], there exists a state  $s' \in S'$  such that  $\text{val}_{\text{OPAR}}^{\mathcal{M}'}(s') = 1$ , and it is easy to see that  $s' \neq \text{trap}$  and thus  $s' \in S \setminus W_2$ . But this implies that  $\text{val}_{\text{OPAR}}^{\mathcal{M}}(s') = 1$  and thus  $s' \in W_2$ , a contradiction.
4. Let  $0 \stackrel{\text{def}}{=} \text{Term}(j) \cup \text{OPAR}$ . For Equation (2), the first inequality  $\text{val}_{0 \cap FW_0}(s) \leq \text{val}_{FW_0}(s)$  is trivial, since  $0 \cap FW_0 \subseteq FW_0$ . To show the reverse inequality, consider the strategy  $\sigma$  that first plays like an optimal MD strategy  $\sigma'$  for the objective  $FW_0$  and after reaching  $W_0$  switches to an almost surely winning MD strategy  $\sigma''$  for the objective  $\text{Loss}$ . Then  $\text{val}_{0 \cap FW_0}(s) \geq \mathcal{P}_{\sigma, s}(0 \cap FW_0) \geq \mathcal{P}_{\sigma, s}(\text{Loss} \cap FW_0) = \mathcal{P}_{\sigma', s}(FW_0) = \text{val}_{FW_0}(s)$ , where the second inequality is due to  $\text{LimInf}(= -\infty) \subseteq \text{Term}(j)$ .
- For Equation (3), the first inequality is again due to the fact that  $\text{LimInf}(= -\infty) \subseteq \text{Term}(j)$  for all  $j \geq 0$ . Towards the second inequality of Equation (3) we have

$$\begin{aligned}
& \text{val}_0(s) \\
&= \sup_{\sigma} \mathcal{P}_{\sigma, s}(0) \\
&= \sup_{\sigma} (\mathcal{P}_{\sigma, s}(0 \cap FW_0) + \mathcal{P}_{\sigma, s}(0 \cap \overline{FW_0})) && \text{(Law of total probability)} \\
&\leq \sup_{\sigma} \mathcal{P}_{\sigma, s}(0 \cap FW_0) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(0 \cap \overline{FW_0}) && (\sup(f+g) \leq \sup f + \sup g) \\
&= \sup_{\sigma} \mathcal{P}_{\sigma, s}(FW_0) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(0 \cap \overline{FW_0}) && \text{(Equation (2))} \\
&\leq \text{val}_{\text{Loss}}(s) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(0 \cap \overline{FW_0}) && \text{(Item 2)}
\end{aligned}$$

We can upper-bound the second summand above as follows.

$$\begin{aligned}
& \sup_{\sigma} \mathcal{P}_{\sigma, s}(0 \cap \overline{FW_0}) \\
&= \sup_{\sigma} \mathcal{P}_{\sigma, s}((\text{Term}(j) \cup \text{OPAR}) \cap \overline{FW_0}) \\
&\leq \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{FW_0}) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{OPAR} \cap \overline{FW_0}) && \text{(Union bound)} \\
&\leq \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{FW_1}) + \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{OPAR} \cap \overline{FW_2}) && \text{(Item 1)} \\
&= \sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{FW_1}) && \text{(Item 3)} \quad \blacktriangleleft
\end{aligned}$$

We show that the term  $\sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{FW_1})$  in Equation (3) can be made arbitrarily small for large  $j$ . To this end, we use [3, Lemma 3.9] (adapted to our notation).

► **Lemma 7** ([3, Lemma 3.9 and Claim 6]). *Let  $\mathcal{M} = (S, S_\square, S_\circ, E, P)$  be a maximizing finite MDP with rewards in unary and  $W_1 \stackrel{\text{def}}{=} \text{AS}(\text{LimInf}(= -\infty))$ . One can compute, in polynomial time, a rational constant  $c < 1$ , and an integer  $h \geq 0$  such that for all  $j \geq h$  and  $s \in S$*

$$\sup_{\sigma} \mathcal{P}_{\sigma, s}(\text{Term}(j) \cap \overline{FW_1}) \leq \frac{c^j}{1-c}.$$

Moreover,  $1/(1-c) \in \mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}))$  and  $h \in \mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}))$ .

► **Lemma 8.** Consider a maximizing MDP  $\mathcal{M} = (S, S_{\square}, S_{\circ}, E, P)$ ,  $\varepsilon > 0$  and the constants  $c, h$  from Lemma 7. For rewards in unary and  $i \geq N$  we have  $\text{val}_{\text{Term}(i) \cup \text{OPAR}}^{\mathcal{M}}(s) - \text{val}_{\text{Loss}}^{\mathcal{M}}(s) \leq \varepsilon$  where  $N \stackrel{\text{def}}{=} \max(h, \lceil \log_c(\varepsilon \cdot (1 - c)) \rceil) \in \mathcal{O}(\exp(|\mathcal{M}|^{\mathcal{O}(1)}) \cdot \log(1/\varepsilon))$ .

For rewards in binary we have  $N \in \mathcal{O}(\exp(\exp(|\mathcal{M}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$ , i.e., the size of  $N$  increases by one exponential.

**Proof sketch.** For rewards in unary, the result follows from Lemma 6 (Equation (3)) and Lemma 7. For rewards in binary, the constants increase by one exponential via encoding binary rewards into unary rewards in a modified MDP. ◀

## 5.2 Computing $N$ for SSGs

In order to compute the bound  $N$  for an SSG  $\mathcal{G}$ , we first consider bounds  $N(s)$  for individual states  $s$  and then take their maximum. Given a state  $s$ , we can use Theorem 5 (Item 3) to obtain an optimal FD strategy (with  $\mathcal{O}(\exp(|\mathcal{G}|^{\mathcal{O}(1)}))$  memory modes)  $\sigma^*(s) = (M, m_0, \text{upd}, \text{nxt})$  for Maximizer from state  $s$  w.r.t. the **Gain** objective. Theorem 5 (Item 1) yields a uniform MD strategy  $\pi^*$  that is optimal for Minimizer from all states  $s$  w.r.t. the **Gain** objective.

► **Lemma 9.** Given an SSG  $\mathcal{G} = (S, (S_{\square}, S_{\circ}, S_{\circ}), E, P)$  and  $\varepsilon > 0$ , we can compute a number  $N \in \mathbb{N}$  such that for all  $i \geq N$  and states  $s \in S$  we have

$$\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) - \varepsilon \leq \text{val}_{\text{Gain}}^{\mathcal{G}}(s) - \varepsilon \leq \inf_{\pi} \mathcal{P}_{\sigma^*(s), \pi, s}^{\mathcal{G}}(\text{EN}(i) \cap \text{EPAR}) \leq \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \quad (4)$$

i.e.,  $\sigma^*(s)$  is  $\varepsilon$ -optimal for Maximizer for  $\text{EN}(i) \cap \text{EPAR}$  for all  $i \geq N$ . In particular,  $0 \leq \text{val}_{\text{Gain}}^{\mathcal{G}}(s) - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon$ .

Moreover,  $\pi^*$  is  $\varepsilon$ -optimal for Minimizer from any state  $s$  for  $i \geq N$ .

$$\sup_{\sigma} \mathcal{P}_{\sigma, \pi^*, s}^{\mathcal{G}}(\text{EN}(i) \cap \text{EPAR}) \leq \sup_{\sigma} \mathcal{P}_{\sigma, \pi^*, s}^{\mathcal{G}}(\text{Gain}) = \text{val}_{\text{Gain}}^{\mathcal{G}}(s) \leq \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) + \varepsilon \quad (5)$$

For rewards in unary,  $N$  is doubly exponential, i.e.,  $N \in \mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  and it can be computed in exponential time. For rewards in binary, the size of  $N$  and its computation time increase by one exponential, respectively.

**Proof.** Assume that rewards are in unary. The first inequality of (4) holds because  $\text{EN}(i) \cap \text{EPAR} \subseteq \text{Gain}$  for any  $i$ . The third inequality of (4) follows from the definition of the value. Towards the second inequality of (4), we consider the minimizing MDP  $\mathcal{M}(s) \stackrel{\text{def}}{=} \mathcal{G}^{\sigma^*(s)}$  obtained by fixing the Maximizer strategy  $\sigma^*(s)$ . Since  $\sigma^*(s)$  is optimal for Maximizer from state  $s$  wrt. the objective **Gain**, Lemma 4 yields that

$$\text{val}_{\text{Gain}}^{\mathcal{G}}(s) = \text{val}_{\text{Gain}}^{\mathcal{M}(s)}((m_0, s)). \quad (6)$$

Since  $\sigma^*(s)$  has  $\mathcal{O}(\exp(|\mathcal{G}|^{\mathcal{O}(1)}))$  memory modes, the size of  $\mathcal{M}(s)$  is exponential in  $|\mathcal{G}|$  and  $\mathcal{M}(s)$  can be computed in exponential time.

Now we consider the dual maximizing MDP  $\mathcal{M}(s)^d$  and the objectives  $\text{Term}(i) \cup \text{OPAR}$  and **Loss**. (Note that  $\mathcal{M}(s)^d$  has the same size as  $\mathcal{M}(s)$ .) From Lemma 8, we obtain a bound  $N(s) \in \mathbb{N}$  such that for all  $i \geq N(s)$

$$0 \leq \text{val}_{\text{Term}(i) \cup \text{OPAR}}^{\mathcal{M}(s)^d}((m_0, s)) - \text{val}_{\text{Loss}}^{\mathcal{M}(s)^d}((m_0, s)) \leq \varepsilon. \quad (7)$$

By Lemma 8 and Lemma 7,  $N(s)$  is exponential in  $|\mathcal{M}(s)^d|$  and thus doubly exponential in  $|\mathcal{G}|$ , i.e.,  $N(s) \in \mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$ . Moreover,  $N(s)$  can be computed in time polynomial in  $|\mathcal{M}(s)^d|$  and thus in time exponential in  $|\mathcal{G}|$ . By duality, we can rewrite Equation (7) for  $\mathcal{M}(s)$  as follows. For all  $i \geq N(s)$

$$0 \leq \text{val}_{\text{Gain}}^{\mathcal{M}(s)}((m_0, s)) - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{M}(s)}((m_0, s)) \leq \varepsilon. \quad (8)$$

In order to get a uniform upper bound that holds for all states, let  $N \stackrel{\text{def}}{=} \max_{s \in S} N(s)$ . Since  $|S|$  is linear, we still have  $N \in \mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  and it can be computed in exponential time in  $|\mathcal{G}|$ . Finally, we can show the second inequality of (4).

$$\begin{aligned} & \inf_{\pi} \mathcal{P}_{\sigma^*(s), \pi, s}^{\mathcal{G}}(\text{EN}(i) \cap \text{EPAR}) \\ &= \inf_{\pi} \mathcal{P}_{\pi, (m_0, s)}^{\mathcal{M}(s)}(\text{EN}(i) \cap \text{EPAR}) \\ &= \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{M}(s)}((m_0, s)) \\ &\geq \text{val}_{\text{Gain}}^{\mathcal{M}(s)}((m_0, s)) - \varepsilon && \text{by } i \geq N \geq N(s) \text{ and Equation (8)} \\ &= \text{val}_{\text{Gain}}^{\mathcal{G}}(s) - \varepsilon && \text{by (6)} \end{aligned}$$

The first inequality of (5) holds because  $\text{EN}(i) \cap \text{EPAR} \subseteq \text{Gain}$  for any  $i$ . The equality in (5) holds by the optimality of  $\pi^*$ . The second inequality of (5) follows from the previously stated consequence of (4).

For rewards in binary, the sizes of the numbers  $N(s)$  (and hence  $N$ ) and the time to compute it increase by one exponential by Lemma 8.  $\blacktriangleleft$

## 6 Unfolding the Game to Energy Level $N$

Given an SSG  $\mathcal{G} = (S, (S_{\square}, S_{\diamond}, S_{\circ}), E, P)$  and error tolerance  $\varepsilon > 0$ , for each state  $s \in S$  and energy level  $i \geq 0$ , we want to compute a rational number  $v'$  which satisfies  $0 \leq v' - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon$ , and  $\varepsilon$ -optimal FD strategies  $\sigma_{\varepsilon}$  and  $\pi_{\varepsilon}$  for Maximizer and Minimizer, resp. We achieve this by constructing a finite-state parity game  $\mathcal{G}'$  that closely approximates the original game  $\mathcal{G}$ , as described in Section 3(Item 3).

For clarity, we explain the construction in two steps. In the first step, we consider a finite-state parity game  $\mathcal{G}[N]$ . (Unlike  $\mathcal{G}'$ , the game  $\mathcal{G}[N]$  is not actually constructed. It just serves as a part of the correctness proof.)  $\mathcal{G}[N]$  encodes the energy level up-to  $N + R$  (where  $R$  is the maximal transition reward) into the states, i.e., it has states of the form  $(s, k)$  with  $k \leq N + R$ . It imitates the original game  $\mathcal{G}$  till energy level  $N + R$ , but at any state  $(s, i)$  with energy level  $i \geq N$  it jumps to a winning state with probability  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$  and to a losing state with probability  $1 - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$ . (We need the margin up-to  $N + R$ , because transitions can have rewards  $> 1$ , so the level  $N$  might not be hit exactly.) Similarly, at states  $(s, 0)$  with energy level 0, we jump to a losing state. The coloring function in the new game  $\mathcal{G}[N]$  derives its colors from the colors in the original game  $\mathcal{G}$ , i.e., all states  $(s, i)$  have the same color as  $s$  in  $\mathcal{G}$ .

By construction of  $\mathcal{G}[N]$ , for  $i \leq N$ , the EPAR value of  $(s, i)$  in  $\mathcal{G}[N]$  coincides with  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$ .

In the second step, since we do not know the exact values  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$  for  $N + R \geq i > N$ , we approximate these by the slightly larger  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$ . I.e., we modify  $\mathcal{G}[N]$  by replacing the probability values  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$  for the jumps to the winning state by  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$ . Let  $\mathcal{G}'$  be the resulting finite-state parity game. It follows from Lemma 9 that  $0 \leq \text{val}_{\text{Gain}}^{\mathcal{G}}(s) - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon$  for  $i \geq N$  and  $\text{Lval}_{\text{EN} \cap \text{EPAR}}^{\mathcal{G}}(s) = \text{val}_{\text{Gain}}^{\mathcal{G}}(s)$ . Thus  $\mathcal{G}'$   $\varepsilon$ -over-approximates  $\mathcal{G}[N]$  and  $\mathcal{G}$ , and we obtain the following lemma.

► **Lemma 10.** For all states  $s$  and all  $0 \leq i \leq N$

$$\begin{aligned} \text{val}_{\text{EPAR}}^{\mathcal{G}[N]}((s, i)) &= \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s), \text{ and} \\ 0 \leq \text{val}_{\text{EPAR}}^{\mathcal{G}'}((s, i)) - \text{val}_{\text{EPAR}}^{\mathcal{G}[N]}((s, i)) &\leq \varepsilon. \end{aligned}$$

Now we are ready to prove the main theorem.

► **Theorem 2.** Let  $\mathcal{G} = (S, (S_{\square}, S_{\diamond}, S_{\circ}), E, P)$  be an SSG with transition rewards in unary assigned by function  $r$  and colors assigned to states by function  $\text{Col}$ . For every state  $s \in S$ , initial energy level  $i \geq 0$  and error margin  $\varepsilon > 0$ , one can compute

1. a rational number  $v'$  such that  $0 \leq v' - \text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s) \leq \varepsilon$  in 2-NEXPTIME.<sup>2</sup>
2.  $\varepsilon$ -optimal FD strategies  $\sigma_{\varepsilon}$  and  $\pi_{\varepsilon}$  for Maximizer and Minimizer, resp., in 2-NEXPTIME. These strategies use  $\mathcal{O}(2\text{-EXP}(|\mathcal{G}|) \cdot \log(\frac{1}{\varepsilon}))$  memory modes.

For rewards in binary, the bounds above increase by one exponential.

**Proof.** For  $i > N$  we output  $v' = \text{val}_{\text{Gain}}^{\mathcal{G}}(s)$ , which satisfies the condition by Lemma 9. For  $i \leq N$  we output  $v' = \text{val}_{\text{EPAR}}^{\mathcal{G}'}((s, i))$ , which satisfies the condition by Lemma 10. By Theorem 5, the values  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  are rational for all states  $s$ . Therefore all probability values in  $\mathcal{G}'$  are rational and thus the EPAR values of all states in  $\mathcal{G}'$  are rational. Hence our numbers  $v'$  are always rational.

By Theorem 5, the values  $\text{val}_{\text{Gain}}^{\mathcal{G}}(s)$  for all states  $s \in S$  can be computed in exponential time. By Lemma 9,  $N \in \mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  is doubly exponential. Therefore, we can construct  $\mathcal{G}'$  in  $\mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  time and space. Questions about the parity values of states in  $\mathcal{G}'$  can be decided in nondeterministic time polynomial in  $|\mathcal{G}'|$ . Thus the numbers  $v'$  are computed in 2-NEXPTIME.

Towards Item 2, we construct  $\varepsilon$ -optimal FD strategies  $\sigma_{\varepsilon}$  for Maximizer (resp.  $\pi_{\varepsilon}$  for Minimizer) for  $\text{EN}(i) \cap \text{EPAR}$  in  $\mathcal{G}$ . Let  $\hat{\sigma}$  (resp.  $\hat{\pi}$ ) be optimal MD strategies for Maximizer (resp. Minimizer) for the objective EPAR in  $\mathcal{G}'$ , which exist by Remark 1. Since these strategies are MD, they can be guessed in nondeterministic time polynomial in the size  $|\mathcal{G}'|$ , and thus in  $\mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  nondeterministic time.

Then  $\sigma_{\varepsilon}$  plays as follows. While the current energy level  $j$  ( $i$  plus the sum of the rewards so far) stays  $< N$ , then, at any state  $s'$ , play like  $\hat{\sigma}$  at state  $(s', j)$  in  $\mathcal{G}'$ . Once the energy level reaches a value  $\geq N$  at some state  $s'$  for the first time, then play like  $\sigma^*(s')$  forever. (Recall that  $\sigma^*(s')$  is the optimal FD Maximizer strategy for Gain from state  $s'$  from Section 5.2.)  $\sigma_{\varepsilon}$  is  $\varepsilon$ -optimal by Lemma 10 and Lemma 9. It needs to remember the energy level up-to  $N$  while simulating  $\hat{\sigma}$ . Moreover,  $\sigma^*(s')$  needs  $\mathcal{O}(\exp(|\mathcal{G}|^{\mathcal{O}(1)}))$  memory modes by Theorem 5. Finally, it needs to remember the switch from  $\hat{\sigma}$  to  $\sigma^*(s')$ . Since  $N \in \mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  dominates the rest,  $\sigma_{\varepsilon}$  uses  $\mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  memory modes.

Similarly,  $\pi_{\varepsilon}$  plays as follows. While the current energy level  $j$  stays  $< N$ , at any state  $s'$ , play like  $\hat{\pi}$  at state  $(s', j)$  in  $\mathcal{G}'$ . Once the energy level reaches a value  $\geq N$  (at any state) for the first time, then play like  $\pi^*$  forever (where  $\pi^*$  is the uniform optimal MD Minimizer strategy for Gain from Section 5.2.)  $\pi_{\varepsilon}$  is  $\varepsilon$ -optimal by Lemma 10 and Lemma 9. While  $\pi^*$  is MD and does not use any memory,  $\pi_{\varepsilon}$  still needs to remember the energy level up-to  $N$  while simulating  $\hat{\pi}$ , and thus it uses  $\mathcal{O}(\exp(\exp(|\mathcal{G}|^{\mathcal{O}(1)})) \cdot \log(1/\varepsilon))$  memory modes.

For rewards in binary, all bounds increase by one exponential via an encoding of  $\mathcal{G}$  into an exponentially larger but equivalent game with rewards in unary. ◀

<sup>2</sup> We write “computing a number  $v'$  in 2-NEXPTIME” as a shorthand for the property that questions like  $v' \leq c$  for constants  $c$  are decidable in 2-NEXPTIME.

No nontrivial lower bounds are known on the computational complexity of approximating  $\text{val}_{\text{EN}(i) \cap \text{EPAR}}^{\mathcal{G}}(s)$ . However, even without the parity part, the problem appears to be hard. The best known algorithm for approximating the value of the energy objective (resp. the dual termination objective) runs in NEXPTIME for SSGs with rewards in unary [3].

As for lower bounds on the strategy complexity,  $\varepsilon$ -optimal Maximizer strategies need at least an exponential number of memory modes (for any  $0 < \varepsilon < 1$ ) even in maximizing MDPs. This can easily be shown by extending the example in Lemma 3 (Item 2) and [24, Fig. 1 and Prop. 4] that shows the lower bound for the Gain objective. First loop in a state with an unfavorable color to accumulate a sufficiently large reward (depending on  $\varepsilon$ ) and then switch to the MDP in [24, Fig. 1 and Prop. 4] to play for Gain (since  $\text{EN}(i) \cap \text{EPAR}$  will be very close to Gain then). Even the latter part requires exponentially many memory modes.

## 7 Conclusion & Extensions

We gave a procedure to compute  $\varepsilon$ -approximations of the value of combined energy-parity objectives in SSGs. The decidability of questions about the exact values is open, but the problem is at least as hard as the positivity problem for linear recurrence sequences [31, Section 5.2.3]. Unlike almost surely winning Maximizer strategies which require infinite memory in general [28, 29],  $\varepsilon$ -optimal strategies for either player require only finite memory with at most doubly exponentially many memory modes.

An interesting topic for further study is whether these results can be extended to other combined objectives where the parity part is replaced by something else, i.e., energy-X for some objective X (e.g., some other color-based condition like Rabin/Streett, or a quantitative objective about multi-dimensional transition rewards). While our proofs are not completely specific to parity, they do use many strong properties that parity satisfies.

- Shift-invariance of EPAR is used in several places, e.g. in Lemma 6 (and thus its consequences) and for the correctness of the constructions in Section 6.
- We use the fact that EPAR goes well together with  $\text{LimInf}(> -\infty)$ , i.e., the objective  $\text{Gain} = \text{LimInf}(> -\infty) \cap \text{EPAR}$  allows optimal FD strategies for Maximizer in MDPs; cf. Lemma 3.
- The submixing property of  $\text{OPAR} = \overline{\text{EPAR}}$  is used in Theorem 5 to lift Lemma 3 from MDPs to SSGs.

---

## References

- 1 Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
- 2 Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.
- 3 T. Brázdil, V. Brožek, K. Etessami, and A. Kučera. Approximating the Termination Value of One-Counter MDPs and Stochastic Games. *Information and Computation*, 222:121–138, 2013. [arXiv:1104.4978](https://arxiv.org/abs/1104.4978).
- 4 T. Brázdil, A. Kučera, and P. Novotný. Optimizing the expected mean payoff in energy Markov decision processes. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 9938 of *LNCS*, pages 32–49, 2016.
- 5 Tomáš Brázdil, Václav Brozek, and Kousha Etessami. One-Counter Stochastic Games. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 108–119, Dagstuhl, Germany, 2010. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. Full version at [arXiv:1009.5636](https://arxiv.org/abs/1009.5636). doi:10.4230/LIPIcs.FSTTCS.2010.108.

- 6 Arindam Chakrabarti, Luca De Alfaro, Thomas A Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133, 2003.
- 7 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6199 of *LNCS*, pages 599–610, 2010.
- 8 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6907, pages 206–218, 2011.
- 9 Krishnendu Chatterjee and Laurent Doyen. Games and Markov decision processes with mean-payoff parity and energy parity objectives. In *Mathematical and Engineering Methods in Computer Science (MEMICS)*, volume 7119 of *LNCS*, pages 37–46. Springer, 2011.
- 10 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Youssef Oualhadj. Perfect-information stochastic mean-payoff parity games. In *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, volume 8412 of *LNCS*, 2014.
- 11 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, volume 4423 of *LNCS*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0\_12.
- 12 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Simple stochastic parity games. In *Computer Science Logic (CSL)*, volume 2803 of *LNCS*, pages 100–113. Springer, 2003.
- 13 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Quantitative stochastic parity games. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 121–130. SIAM, 2004.
- 14 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.
- 15 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.
- 16 Laure Daviaud, Martin Jurdziński, and Ranko Lazić. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In *Logic in Computer Science (LICS)*, pages 325–334, 2018.
- 17 Luca De Alfaro and Thomas A Henzinger. Interface automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001.
- 18 David L. Dill. *Trace theory for automatic hierarchical verification of speed-independent circuits*, volume 24. MIT press Cambridge, 1989.
- 19 Graham Everest, Alfred Jacobus van der Poorten, Igor Shparlinski, and Thomas Ward. *Recurrence sequences*. ACM, 2003.
- 20 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.
- 21 Dean Gillette. Stochastic games with zero stop probabilities. *Contributions to the Theory of Games*, 3:179–187, 1957.
- 22 Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 847–862, 2010. URL: <http://epubs.siam.org/doi/abs/10.1137/1.9781611973075.69>.
- 23 Hugo Gimbert and Edon Kelmendi. Submixing and shift-invariant stochastic games. working paper or preprint, 2022. arXiv:1401.6575.
- 24 Hugo Gimbert, Youssef Oualhadj, and Soumya Paul. Computing optimal strategies for Markov decision processes with parity and positive-average conditions. working paper or preprint, 2011. URL: <https://hal.science/hal-00559173/en/>.
- 25 Marcin Jurdziński. Deciding the winner in parity games is in  $UP \cap co-UP$ . *Information Processing Letters*, 68(3):119–124, 1998.
- 26 A. Maitra and W. Sudderth. Stochastic games with Borel payoffs. In *Stochastic Games and Applications*, pages 367–373. Kluwer, Dordrecht, 2003.
- 27 Donald A. Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

- 28 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. MDPs with Energy-Parity Objectives. In *Logic in Computer Science (LICS)*. IEEE, 2017. [arXiv:1701.02546](#).
- 29 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. Simple stochastic games with almost-sure energy-parity objectives are in NP and coNP. In *Proc. of Fossacs*, volume 12650 of *LNCS*, 2021. Extended version on arXiv. [arXiv:2101.06989](#).
- 30 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, 2015.
- 31 Jakob Piribauer. *On non-classical stochastic shortest path problems*. PhD thesis, Technische Universität Dresden, Germany, 2021. URL: <https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-762812>.
- 32 Jakob Piribauer and Christel Baier. On Skolem-Hardness and Saturation Points in Markov Decision Processes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of ICALP*, volume 168 of *LIPICs*, pages 138:1–138:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. [doi:10.4230/LIPICs.ICALP.2020.138](#).
- 33 Jakob Piribauer and Christel Baier. Positivity-hardness results on Markov decision processes. working paper or preprint, 2023. [arXiv:2302.13675](#).
- 34 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Annual Symposium on Principles of Programming Languages (POPL)*, pages 179–190, 1989.
- 35 Peter J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.
- 36 Lloyd S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.
- 37 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.



# Dynamic Planar Embedding Is in DynFO

Samir Datta  

Chennai Mathematical Institute & UMI ReLaX, Chennai, India

Asif Khan 

Chennai Mathematical Institute, India

Anish Mukherjee  

University of Warwick, Coventry, UK

---

## Abstract

Planar Embedding is a drawing of a graph on the plane such that the edges do not intersect each other except at the vertices. We know that testing the planarity of a graph and computing its embedding (if it exists), can efficiently be computed, both sequentially [24] and in parallel [33], when the entire graph is presented as input.

In the dynamic setting, the input graph changes one edge at a time through insertion and deletions and planarity testing/embedding has to be updated after every change. By storing auxiliary information we can improve the complexity of dynamic planarity testing/embedding over the obvious recomputation from scratch. In the sequential dynamic setting, there has been a series of works [17, 25, 20, 22], culminating in the breakthrough result of  $\text{polylog}(n)$  sequential time (amortized) planarity testing algorithm of Holm and Rotenberg [21].

In this paper we study planar embedding through the lens of DynFO, a parallel dynamic complexity class introduced by Patnaik et al [31] (also [16]). We show that it is possible to dynamically maintain whether an edge can be inserted to a planar graph without causing non-planarity in DynFO. We extend this to show how to maintain an embedding of a planar graph under both edge insertions and deletions, while rejecting edge insertions that violate planarity.

Our main idea is to maintain embeddings of only the triconnected components and a special two-colouring of separating pairs that enables us to side-step cascading flips when embedding of a biconnected planar graph changes, a major issue for sequential dynamic algorithms [22, 21].

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Complexity theory and logic; Theory of computation  $\rightarrow$  Finite Model Theory

**Keywords and phrases** Dynamic Complexity, Planar graphs, Planar embedding

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.39

**Related Version** *Full Version:* <https://arxiv.org/abs/2307.09473>

**Funding** *Samir Datta:* Partially funded by a grant from Infosys foundation.

*Asif Khan:* Partially funded by a grant from Infosys foundation.

*Anish Mukherjee:* Research supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), by EPSRC award EP/V01305X/1.

**Acknowledgements** Thanks to Nils Vortmeier and Thomas Zeume for illuminating discussions.

## 1 Introduction

Planar graphs are graphs for which there exists an embedding of vertices on the plane such that the edges can be drawn without intersecting with each other, except at their endpoints. The notion of planar graphs is fundamental to graph theory as underlined by the Kuratowski theorem [26]. The planarity testing problem is to determine if the encoded graph is planar and the planar embedding problem is to construct such an embedding. These are equally fundamental questions to computer science and their importance has been recognized from the early 1970s in the linear time algorithm by Hopcroft and Tarjan [24]. Since then there have



© Samir Datta, Asif Khan, and Anish Mukherjee;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 39; pp. 39:1–39:15

Leibniz International Proceedings in Informatics



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

been a plethora of algorithmic solutions presented for the planarity testing and embedding problems such as [28, 4, 18] that culminated in an alternative linear-time algorithm [18], a work efficient parallel algorithm running in  $O(\log n)$  time [33], a deterministic logspace algorithm [1, 13], and many more.

All of the above algorithms are static i.e., the input is presented at once and we need to answer the planarity testing query and produce an embedding only once. However, in many real-life scenarios, the input is itself dynamic and evolves by insertion and deletion of edges. The same query can be asked at any instance and even the embedding may be required. Rather than recomputing the result from scratch after every update to the input in many scenarios, it is advantageous to preserve some auxiliary data such that each testing or embedding query can be answered much faster than recomputation from scratch. The notion of “fast” can be quantified via the sequential time required to handle the updates and queries which can be achieved in polylogarithmic time as in the recent breakthrough works of [22, 21]. These in turn built upon the previous work that dealt with only a partially dynamic model of computation – insertion only [32, 3, 34] or, deletion only [25] or the fully dynamic model (that supports both insertions and deletions) but with polynomial time updates [17].

Our metric for evaluating updates is somewhat different and determined according to the Dynamic Complexity framework of Immerman and Patnaik [31], and is closely related to the setting of Dong, Su, and Topor [16]. In it, a dynamic problem is characterised by the smallest complexity class in which it is possible to place the updates to the auxiliary database and still be able to answer the queries (notice that if the number of possible queries is polynomial we can just maintain the answers of all queries in the auxiliary database).

Notable amongst these has been the first-order logic formulas or equivalently, the descriptive complexity class FO. Thus we obtain the class DynFO of dynamic problems for which the updates to the auxiliary data structure are in FO given the input structure and stored auxiliary data structures. The motivation to use first-order logic as the update method has connections to other areas as well e.g., it implies that such queries are highly parallelisable, i.e., can be updated by polynomial-size circuits in constant-time due to the correspondence between FO and uniform  $AC^0$  circuits [2]. From the perspective of database theory, such a program can be translated into equivalent SQL queries.

A particular recent success story of dynamic complexity has been that directed graph reachability (which is provably not in FO) can be maintained in DynFO [8] resolving a conjecture from Immerman and Patnaik [31], open since the inception of the field. Since then, progress has been made in terms of the size of batch updates (i.e., multiple simultaneous insertions and deletions) that can be handled for reachability, distance, and maximum matching [12, 30]. Later, improved bounds have been achieved for these problems in various special graph classes, including in planar graphs [9, 5]. Problems in planar graphs have been studied in the area of dynamic complexity starting much earlier e.g., before the reachability conjecture was resolved, it was shown in [6] that reachability in embedded planar graphs is in DynFO. Also, in [29] it was shown that 3-connected planar graph isomorphism too is in DynFO with some precomputation. However, despite these works the dynamic complexity of the planarity testing problem itself is not yet resolved, let alone maintaining a planar embedding efficiently.

**Our contribution.** In this paper, we build on past work in dynamic complexity to show that a planar embedding can be maintained efficiently, where we test for planarity at every step. Here, by planar embedding we mean a cyclic order on the neighbours of every vertex in some drawing of the graph on the plane (also known as combinatorial embedding [15]).

► **Theorem 1.** *Given a dynamic graph undergoing insertion and deletion of edges we can maintain a planar embedding of the graph in DynFO (while never allowing insertion of edges that cause the graph to become non-planar).*

**Organization.** We start with preliminaries concerning graph theory and dynamic complexity in Section 2. We present a technical overview of our work in Section 3. In Section 4 we develop the graph theoretic machinery we need for our algorithm. Next, we describe the dynamic planar embedding algorithm in Section 5. For the implementation details please refer to the full version of this paper [7].

## 2 Preliminaries

We start with some notations followed by graph theoretic preliminaries related to connectivity and planarity – see [14, Chapters 3, 4] for a thorough introduction. Then we reproduce some essentials of Dynamic Complexity from [9, 12].

Given a graph  $G = (V, E)$ , we write  $V(G)$  and  $E(G)$  to denote the sets of vertices and edges of  $G$ , respectively. For a set of edges  $S \subseteq E(G)$  we denote by  $G - S$ , the graph with the edges in  $S$  deleted. Similarly for  $S \subseteq V(G) \times V(G)$  we denote by  $G + S$  the graph to which new edges in  $S$  have been added. For a set of vertices  $T \subseteq V(G)$ , by  $G - T$  we refer to the induced graph  $G[V(G) \setminus T]$ . An undirected path between  $u$  and  $v$  is denoted by  $u \rightsquigarrow v$ .

**Biconnected and Triconnected Decomposition.** We assume familiarity with common connectivity related terminology including 2-vertex connectivity, 3-vertex connectivity and the related separating sets viz. cut vertices, separating pairs and the notion of virtual edges in the triconnected decomposition [23]. Please see the definitions of these terms in the full version [7, Section 2.1]. The following are two data structures that help in representing tree decompositions associated with biconnectivity and 3-connectivity respectively.

1. **BC-tree** or block-cut tree of a connected component of the graph, say  $H$ , denoted by  $T_2(H)$ . The nodes of the tree are the biconnected components (block nodes) and the cut vertices (cut nodes) of  $H$  and the edges are only between cut and block nodes. Block nodes are denoted by  $B$  and the cut nodes are denoted by  $C$ .
2. **SPQR-tree** or the triconnected decomposition tree of a biconnected component of the graph say  $B$ , is denoted by  $T_3(B)$ . The nodes in the SPQR-tree are of one of four types:  $S$  denotes a cycle component (serial node),  $P$  denotes a 3-connected separating pair (parallel node),  $Q$  denotes that there is just a single edge in  $B$ , and  $R$  denotes the 3-connected components or the so-called rigid nodes. There is an edge between an R-node, say  $R_i$  and a P-node, say  $P_j$  if  $V(P_j) \subset V(R_i)$ , and similarly, edges between S and P-nodes are defined.

We will conflate a node in one of the two trees with the corresponding subgraph. For example, an R-node interchangeably refers to the tree node as well the associated rigid subgraph.

**Planar Embedding.** A planar embedding of a graph  $G = (V, E)$  is a mapping of vertices and edges in the plane  $\mathbb{R}^2$  such that the vertices are mapped to distinct points in the plane and every edge is mapped to an arc between the points corresponding to the two vertices incident on it such that no two arcs have any point in common except at their endpoints. This embedding is called a *topological embedding*. Corresponding to a given topological embedding, the faces of the graph are the open regions in  $\mathbb{R}^2 \setminus G$  (plane with points corresponding to the vertices and edges removed), call the set of faces as  $F$ . For a face  $f \in F$ , the set of all the

vertices that lie on the boundary of  $f$ , is denoted by  $V(f)$ . The unbounded face is called the outer face. An embedding on the surface of a sphere is similarly defined. On the sphere, every face is bounded. Two topological embeddings are equivalent if, for every vertex, the cyclic order of its neighbours around the vertex is the same in both embeddings. So, the cyclic order (or rotation scheme) around each vertex defines an equivalence on the topological embeddings. The vertex rotation scheme around each vertex encodes the embedding equivalence class (combinatorial embedding). We now recall two important results. The first one is due to Whitney [35] that says that a 3-connected planar graph has unique planar embedding on the sphere (up to reflection). The second one is due to Mac Lane [27] that shows a biconnected graph is planar iff its triconnected components are planar.

## 2.1 Dynamic Complexity

The goal of a dynamic program is to answer a given query on an *input structure* subject to changes that insert or delete tuples. The program may use an auxiliary data structure represented by an *auxiliary structure* over the same domain. Initially, both input and auxiliary structure are empty; and the domain is fixed during each run of the program.

For a (relational) structure  $\mathcal{I}$  over domain  $D$  and schema  $\sigma$ , a change  $\Delta\mathcal{I}$  consists of sets  $R^+$  and  $R^-$  of tuples for each relation symbol  $R \in \sigma$ . The result  $\mathcal{I} + \Delta\mathcal{I}$  of an application of the change  $\Delta\mathcal{I}$  to  $\mathcal{I}$  is the input structure where  $R^{\mathcal{I}}$  is changed to  $(R^{\mathcal{I}} \cup R^+) \setminus R^-$ . The *size* of  $\Delta\mathcal{I}$  is the total number of tuples in relations  $R^+$  and  $R^-$  and the set of *affected elements* is the (active) domain of tuples in  $\Delta\mathcal{I}$ .

**Dynamic Programs and Maintenance of Queries.** A dynamic program consists of a set of update rules that specify how auxiliary relations are updated after changing the input structure. An *update rule* for updating an  $\ell$ -ary auxiliary relation  $T$  after a change is a first-order formula  $\varphi$  over schema  $\tau \cup \tau_{\text{aux}}$  with  $\ell$  free variables, where  $\tau_{\text{aux}}$  is the schema of the auxiliary structure. After a change  $\Delta\mathcal{I}$ , the new version of  $T$  is  $T := \{\bar{a} \mid (\mathcal{I} + \Delta\mathcal{I}, \mathcal{A}) \models \varphi(\bar{a})\}$  where  $\mathcal{I}$  is the old input structure and  $\mathcal{A}$  is the current auxiliary structure. Note that a dynamic program can choose to have access to the old input structure by storing it in its auxiliary relations.

For a state  $\mathcal{S} = (\mathcal{I}, \mathcal{A})$  of the dynamic program  $\mathcal{P}$  with input structure  $\mathcal{I}$  and auxiliary structure  $\mathcal{A}$ , we denote by  $\mathcal{P}_\alpha(\mathcal{S})$ , the state of the program after applying a change sequence  $\alpha$  and updating the auxiliary relations accordingly. The dynamic program *maintains* a  $q$ -ary query  $Q$  under changes that affect  $k$  elements (under changes of size  $k$ , respectively) if it has a  $q$ -ary auxiliary relation *Ans* that at each point stores the result of  $Q$  applied to the current input structure. More precisely, for each non-empty sequence  $\alpha$  of changes that affect  $k$  elements (changes of size  $k$ , respectively), the relation *Ans* in  $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$  and  $Q(\alpha(\mathcal{I}_\emptyset))$  coincide, where  $\mathcal{I}_\emptyset$  is an empty input structure,  $\mathcal{S}_\emptyset$  is the auxiliary structure with empty auxiliary relations over the domain of  $\mathcal{I}_\emptyset$ , and  $\alpha(\mathcal{I}_\emptyset)$  is the input structure after applying  $\alpha$ . If a dynamic program maintains a query, we say that the query is in DynFO.

## 3 Technical Overview

It is well known from Whitney's theorem [35] that 3-connected planar graphs are rigid i.e., they (essentially) have a unique embedding. Thus, for example, under the promise that the graph remains 3-connected and planar it is easy to maintain an embedding in DynFO (see for example [29]). An edge insertion occurs within a face and there are only local changes to the embedding – restricted to a face. Deletions are exactly the reverse.

On the other extreme are trees, which are minimally connected. These are easy to maintain as well because any vertex rotation scheme is realisable. However, biconnected components are not rigid and yet not every rotation scheme for a vertex is valid (see Figure 3a for an illustration). The real challenge is in maintaining embeddings of biconnected components.

This has been dealt with in literature by decomposing biconnected graphs into 3-connected components (which are rigid components in the context of planar graphs). The 3-connected components are organized into trees<sup>1</sup> popularly called SPQR-trees [3]. The approach is to use the rigidity of the 3-connected planar components and the flexibility of trees to maintain a planar embedding of biconnected graphs. In order to maintain a planar embedding of connected graphs we need a further tree decomposition into biconnected components that yields the so-called block-cut trees or BC-trees ([14, Lemma 3.1.4], [19]). Notice that the tree decomposition into SPQR-trees and BC-trees is Logspace hard [10] and hence not in FO. Thus, in the parallel dynamic setting, we emulate previous sequential dynamic algorithms in *maintaining* (rather than computing from scratch) SPQR-trees and BC-trees in our algorithm.

**Issues with biconnected embedding.** The basic problem with maintaining biconnected planar components is their lack of rigidity (with reaching complete flexibility). Thus insertion of an edge into a biconnected component might necessitate changing the embedding through operations called *flips* and *slides* in literature [22, 21] (see [7, Figure 4]). We might need lots of flips and slides for a single edge insertion – causing an exponentially large search space. We now proceed to describe these changes in more detail.

In the simplest form consider a biconnected graph and a separating pair contained within, separating the component into two 3-connected components. We can reflect one of the 3-connected components, that is, the vertex rotation for each vertex in the 3-connected component is reversed. More intuitively, mirror a piece across one of its separating pairs. In more complicated cases there may be cascading flips i.e., reflections across multiple separating pairs in the biconnected component. We also need to deal with *slides*, that is changes in ordering of the biconnected components at a separating pair. A single edge insertion might need multiple flips and slides.

This induces the definition of flip-distance i.e., the minimum number of flips and slides to change one embedding of the graph to another. Intuitively, the flip-distance lower bounds the sequential time needed to transition from one embedding to another.

Thus a crucial part of previous algorithms [22] deals with maintaining an embedding of small flip distance with every possible embedding that can arise after a single change. In [21] this algorithm is converted to a fully dynamic algorithm that handles updates in  $O(\log^3 n)$  time using a sophisticated amortization over the number of flips required to transition to an appropriate embedding dominates the running time of their algorithm. Notice that [22] handles changes in  $O(\log^3 n)$  worst case time but only in the incremental setting.

**Our approach for dynamic planarity testing.** We now switch to motivating our approach in the parallel dynamic setting that is fully dynamic and does not use amortization. There are fundamentally two issues to be resolved while inserting an edge – one is whether the resulting graph is planar. The other is, how to update the embedding, possibly by performing multiple flips, when the graph remains planar. Let us first focus on biconnected graphs and the corresponding tree decomposition SPQR-tree introduced in Section 2.

---

<sup>1</sup> The tree decomposition of a biconnected graph into 3-connected pieces is a usual tree decomposition ([14, Chapter 12.3]).

To check for planarity on insertion of an edge, we introduce the notion of  $P_i, P_j$ -coherent paths. A path between two P-nodes  $P_i, P_j$  in an SPQR-tree is said to be coherent if for every R-node  $R_k$  on the path, all the (at most four) vertices of the two adjacent P-nodes are all on one face of  $R_k$  (see Lemma 4). This yields a combinatorial characterisation of coherent paths, given an embedding of each rigid component. The embedding of a rigid component is dealt with separately, later on. The significance of coherent paths stems from a crucial lemma (Lemma 4). This shows that an edge  $(a, b)$  is insertable in the graph preserving planarity if and only if the “projection” of any simple  $a, b$  path in the graph<sup>2</sup> onto the corresponding SPQR-tree roughly corresponds to a coherent path. This yields a criterion for testing planarity after an edge insertion which can be implemented in FO.

**Insertions in biconnected components.** Having filtered out non-planarity causing edges we turn to the question of how to construct the new planar embedding of the biconnected components after an edge insertion. The answer will lead us to investigate how to embed a rigid component when it is synthesized from a biconnected component.

It is in this context that we introduce the notion of two-colouring of separating pairs. This is a partial sketch of the new 3-connected component formed after an edge insertion. More concretely, the separating pairs along the path in the SPQR-tree are no longer separating pairs after the edge insertion and the common face (as ensured by the crucial lemma alluded to above) on which all the endpoints of the previous separating pairs lie splits into two faces. Since the embedding of a 3-connected planar graph is unique, after the edge insertion the two new faces formed are also unique, i.e., do not depend on the embedding. Thus the endpoints of each previous separating pair can be two coloured depending on which of the two faces a separating pair belongs to. We prove in the two-colouring lemma (Lemma 5) that no separating pair has both vertices coloured with the same colour.

Notice that when an edge is inserted such that its endpoints lie on a coherent path, all the rigid components on the path coalesce into one large rigid component (see Figure 3c). Two-colouring allows us to deal with flips by telling us the correct orientation of the coalescing rigid components on edge insertion. This, in turn, allows us to obtain the face-vertex rotation scheme of the modified component. In addition, it helps us to maintain the vertex rotation scheme in some corner cases (when two or more separating pairs share a vertex).

**Face-vertex rotation scheme.** The sceptical reader might question the necessity of maintaining the face-vertex rotation scheme for a 3-connected component. This is necessary for two reasons – first, to apply the planarity test we need to determine the existence of a common face containing a 4-tuple (or 3-tuple) of vertices. The presence of a face-vertex rotation scheme directly shows that this part is in FO. Second and more crucially, we need it to check if a particular triconnected component needs to be reflected after cascaded flips. Maintaining the vertex rotation scheme for biconnected components is now simple – we just need to collate the vertex rotation schemes for individual rigid components into one for the entire graph.

**Handling deletions.** On deleting edges while it’s not necessary to perform additional flips, the rest of the updates is roughly the reverse of insertion. On deleting an edge from a rigid component, we infer two-colourings from the embedding of erstwhile rigid components

---

<sup>2</sup> which satisfies a minimality condition – it does not pass through both vertices of a separating pair

that decompose into pieces. Further, we have to update the coherent paths since possibly more edges are insertable preserving planarity. Notice that when an edge is deleted from a biconnected component this might lead to many simultaneous virtual edge deletions that might in turn cause triconnected components to decompose. Many ( $O(n)$ ) invocations of the above triconnected edge deletion will be needed, but they can be handled in constant parallel time because they independent of each other as far as the updates required are concerned (see Figure 2).

**Extension to the entire embedding.** BC-trees for connected components have blocks and cut vertices as their nodes. We can maintain an embedding for the graph corresponding to a block or B-node as above. Since a non-cut vertex belongs to precisely one block, we can inherit the rotation scheme for such vertices from that of the blocks. For cut vertices, we need to splice together the vertex rotation scheme from each block that the cut vertex is incident on as long as the order respects the ordering provided by individual blocks.

**Low level details of the information maintained.** We maintain BC-tree for each connected component and SPQR-trees for each biconnected component thereof. In each of these trees we maintain betweenness information, i.e., for any three nodes  $X_1, X_2$  and  $X_3$  whether  $X_2$  occurs on the tree path between  $X_1$  and  $X_3$ . We also maintain a two-colouring of separating pairs for each  $P_i, P_j$ -coherent path in every SPQR-tree. For each rigid component  $R_i$  and each cycle component  $S_j$  we maintain their extended planar embedding. Specifically, we maintain the vertex rotation scheme in the following form. For every vertex  $v$ , we maintain triplet(s)  $(v_i, v_j, v_k)$  of neighbours of  $v$  that occur in the clockwise order though not necessarily consecutively. This enables us to insert and delete an arbitrarily large number of neighbours in FO making it crucial for the planar embedding procedure. This would not be possible if we were to handle individual insertions and deletions separately. See [7, Figure 6] for an example. We use a similar representation for the face-vertex rotation scheme.

For biconnected components, we maintain only a planar embedding (not the extended version) since the face-vertex rotation scheme is not necessary.

**Comparison with existing literature.** The main idea behind recent algorithms for planar embedding in the sequential dynamic setting has been optimizing the number of flips necessitated by the insertion of an edge. This uses either a purely incremental algorithm or alternatively, a fully dynamic but amortized algorithm. Since our model of computation is fully dynamic and does not allow for amortization, each change must be handled (i.e., finding out the correct cascading flips) in worst case  $O(1)$ -time on CRCW-PRAM. We note that filtering out edges that violate planarity in dynamic sequential  $t(n)$  time (a *test-and-reject* model) implies an amortized planarity testing algorithm with  $O(t(n))$  time (i.e., a *promise-free* model). In contrast, although we have a test-and-reject model we are unable to relax the model to promise-free because of lack of amortization.

There are weaker promise models such as the one adopted in [11] where for maintaining a bounded tree-width decomposition it is assumed that the graph has tree-width at most  $k$  without validating the promise at every step. In contrast our algorithm can verify the promise that no non-planarity-causing edge is added.

In terms of query model support, most previous algorithms [22, 21, 25] only maintain the vertex rotation scheme in terms of clockwise next neighbour, in fact, [22, 21] need  $O(\log n)$  time to figure out the next neighbour. In contrast, we maintain more information in terms of arbitrary triplets of neighbours in (not necessarily consecutive) clockwise order. This allows

us to sidestep following arbitrarily many pointers, which is not in FO. Finally, in terms of parallel time our algorithm (since it uses  $O(1)$  time per query/update on CRCW-PRAMs) is optimal in our chosen model. In contrast, the algorithm of [21] comes close but fails to achieve the lower bound (of  $\Omega(\log n)$ ) in the sequential model of dynamic algorithms.

#### 4 Graph Theoretic Machinery

In this section, we present some graph theoretic results which will be crucial for our maintenance algorithm. We begin with a simple observation and go on to present some criteria for the planarity of the graph on edge insertion based on the type of the inserted edge.

► **Observation 2.** *For a 3-connected planar graph  $G$ , two planar embeddings  $\mathcal{E}_1$  and  $\mathcal{E}_2$  in the plane have the same vertex rotation scheme if between the two embeddings only the clockwise order of vertices on the boundary of the outer faces of the two embeddings are reverse if each other and all the clockwise order of vertices on the boundary of internal faces is same.*

Notice that due to the above fact, given a planar embedding with its outer face  $F_0$  and an internal face  $F_1$  specified, we can modify it to make  $F_1$  the outer face while keeping the vertex rotation scheme unchanged by just reversing the orientation of the faces  $F_0$  and  $F_1$ .

Next, we present some criteria to determine if an edge to be inserted in a planar graph causes it to become non-planar.

► **Lemma 3.** *For any 3-connected planar graph  $G$ ,  $G + \{a, b\}$  is planar if and only if  $a$  and  $b$  lie on the boundary of a common face.*

Next, let us consider the case where the vertices  $a$  and  $b$  are in the same block, say  $B_i$ , of a connected component of the graph. Let  $R_a, R_b \in V(T_3(B_i))$  be two R-nodes in the SPQR-tree of  $B_i$  such that  $a \in V(R_a)$  and  $b \in V(R_b)$ . Consider the path between  $R_a$  and  $R_b$  nodes in  $T_3(B_i)$ ,  $R_a, P_1, R_1, P_2, \dots, R_k, P_{k+1}, R_b$ , where  $R_i$  and  $P_i$  are R-nodes and S-nodes in  $T_3(B_i)$  respectively, that appear on the path between  $R_a, R_b$  (see Figure 3). We have the following lemma (see the full version [7, Section 8] for the proof).

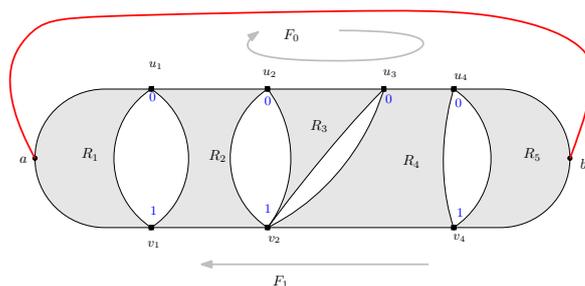
► **Lemma 4.**  *$G + \{a, b\}$  is planar if and only if*

- (a) *all vertices in  $\{a\} \cup V(P_1)$  lie on a common face boundary in the embedding of  $R_a$*
- (b) *all vertices in  $V(P_{k+1}) \cup \{b\}$  lie on a common face boundary in the embedding of  $R_b$ , and*
- (c) *for each  $i \in [k]$  all vertices in  $V(P_i) \cup V(P_{i+1})$  lie on a common face in the embedding of  $R_i$ . Equivalently  $R_a \leftrightarrow R_b$  tree path is a coherent path.*

Consider the case in which  $[2 \xrightarrow{\pm} 3]$  edge  $\{a, b\}$  can be inserted into  $G$  preserving planarity. Notice that the 3-connected components  $R_a, R_1, \dots, R_k, R_b$  coalesce into one 3-connected component after the insertion of the edge. Let this coalesced 3-connected component be  $R_{ab}$ . Obviously,  $\{a, b\}$  would be at the boundary of exactly two faces of  $R_{ab}$ , say  $F_0, F_1$ . We claim that the separating pair vertices in  $\bigcup_{i \in [k+1]} V(P_i)$  all lie either on the boundary of  $F_0$  or  $F_1$ . See Figure 1. We provide the proof of the following lemma in the full version of this paper [7, Section 8].

► **Lemma 5.** *The faces  $F_0$  and  $F_1$  define a partition into two parts on the set of vertices in the separating pairs  $\bigcup_{i \in [k+1]} V(P_i)$  such that, for all  $i \in [k+1]$  the two vertices in  $V(P_i)$  belong to different blocks of the partition.*

Finally, if the vertices  $a$  and  $b$  are in the same connected component but not in the same biconnected component then we use the following lemma to test for edge insertion validity. Let the vertices  $a, b$  lie in a connected component  $C_i$ . Let  $B_a, B_b \in N(T_2(C_i))$  be two block



■ **Figure 1** Two-colouring of P-node vertices.

nodes in the BC-Tree of the connected component  $C_i$  such that  $a \in V(B_a)$  and  $b \in V(B_b)$ . Consider the following path between  $B_a$  and  $B_b$  in  $T_2(C_i)$ ,  $B_a, c_1, B_1, c_2, \dots, c_k, B_k, c_{k+1}, B_b$  where  $B_i$  and  $c_i$  are block and cut nodes respectively, in  $T_2(C_i)$  that appear on the path between  $B_a$  and  $B_b$  (see Figure 2). We abuse the names of cut nodes to also denote the cut vertex's name. Insertion of such an edge, i.e.  $[1 \xrightarrow{+} 2]$  leads to the blocks  $B_a, B_1, \dots, B_k, B_b$  coalescing into one block, call it  $B_{ab}$ . In the triconnected decomposition of  $B_{ab}$  a new cycle component is introduced that consists of the edge  $\{a, b\}$  and virtual edges between the consecutive cut vertices  $c_i, c_{i+1}$ ,  $i \in [k]$ . See Figure 2d. We provide the proof of the following lemma in the full version of this paper [7, Section 8].

► **Lemma 6.**  $G + \{a, b\}$  is planar if and only if (a)  $G[V(B_a)] + \{a, c_1\}$  is planar, (b)  $G[V(B_b)] + \{c_{k+1}, b\}$  is planar, and (c) for each  $i \in [k]$ ,  $G[V(B_i)] + \{c_i, c_{i+1}\}$  is planar.

## 5 Dynamic Planar Embedding

Our idea is to maintain planar embeddings of all triconnected components (S and R-nodes) of the graph and use those to find the embedding of the entire graph. Insertions and deletions of edges change the triconnected components of the graph, i.e., a triconnected component might decompose into multiple triconnected components or multiple triconnected components may coalesce together to form a single one. The same is true of biconnected components, i.e., a biconnected component might decompose into multiple biconnected components or multiple triconnected components may coalesce together to form a single biconnected component.

We discuss here how we update the embeddings of the triconnected components under insertions and deletions, assuming that we have the SPQR-tree and BC-tree relations available at every step (which we show how to maintain in the full version of this paper [7, Section 6]).

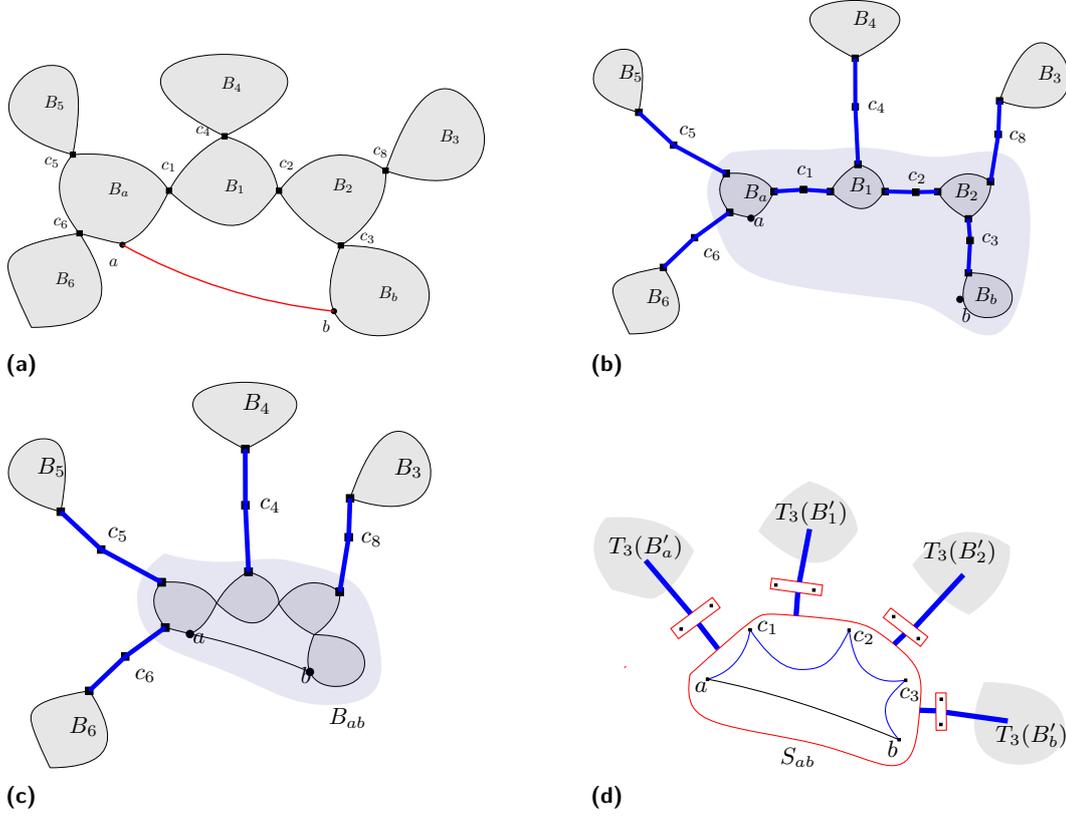
Some of the edge insertions/deletions are easier to describe, for example if the edge is being inserted in a rigid component then only the embedding of that rigid component has to change to reflect the presence of the new edge and introduction of two new faces. Thus, we first establish some notation to differentiate between classes of edges for ease of exposition.

► **Definition 7.** A graph is actually  $i$ -connected if it is  $i$ -connected but is not  $i+1$ -connected for  $i \in \{0, 1, 2\}$ . For  $i = 3$ , a graph is actually  $i$ -connected if the graph is 3-connected.

► **Definition 8.** The type of an edge is  $[i \xrightarrow{\sigma} j]$  where  $i, j \in \{0, 1, 2, 3\}$  and  $\sigma \in \{+, -\}$  such that

- $\sigma = +$  if the edge is being inserted into  $G$ .  $\sigma = -$  if the edge is being deleted.
- both the endpoints are in a common actually  $i$ -connected component before the change and in a common actually  $j$ -connected component after the change.

In the next two subsections we outline the updates in the triconnected planar embedding relations as well as the two-colouring relations which are described in complete detail in the full version of this paper [7, Section 10 and 11].



**Figure 2**  $[1 \xrightarrow{+} 2]$  edge  $\{a, b\}$  insertion. (2a) To-be-inserted edge is highlighted in red. (2b) The path between block nodes  $B_a$  and  $B_b$  is highlighted in the BC-tree. (2c) The BC-tree of the connected component  $CC_i$ .  $B_a \rightsquigarrow B_b$  path has shrunk to a single node  $B_{ab}$  in the BC-tree. (2d) The SPQR-tree of the block  $B_{ab}$  contains a cycle component  $S_{ab}$  made of the inserted edge and the new virtual edges that are introduced after its insertion.

### 5.1 Edge insertion

Find the type  $[i \xrightarrow{+} j]$  of the inserted edge  $\{a, b\}$  (where  $i, j \in \{0, 1, 2, 3\}$ ). This can be done using [7, Lemma 10 and 11]. Depending on the type we branch to one of the following options:

- (a)  $[0 \xrightarrow{+} 1]$  This case affects only the BC-tree and SPQR-tree relations. Embedding and colouring relations remain unaffected.
- (b)  $[1 \xrightarrow{+} 2]$ : In this case, both the endpoints  $a, b$  are in the same connected component  $C$  but not in the same biconnected component  $B_a, B_b$  (see Figure 2a). On this insertion, the BC-tree of the connected component changes. All the biconnected components on the  $B_a \rightsquigarrow B_b$  path in the BC-tree of  $C$  coalesce into one biconnected component  $B_{ab}$  (see Figure 2c). In the BC-tree of the connected component, for each pair of consecutive cut-vertices on the  $B_a \rightsquigarrow B_b$  path, a virtual edge is inserted in the biconnected component shared by the cut-vertex pair. This edge is a  $[2 \xrightarrow{+} 3]$  or  $[3 \xrightarrow{+} 3]$  edge and is handled

below. Since the biconnected components involved are distinct, all these edges can be simultaneously inserted. In addition, a cycle component is introduced for which the face-vertex rotation scheme is computed via the betweenness relation in the BC-tree (see Figure 2d). The two-colouring relation is updated by extending the colouring of old biconnected components across the new cycle component (in Figure 2d, a coherent path from a P-node in  $T_3(B'_a)$  to a P-node in  $T_3(B'_b)$  will have to go via an S-node  $S_{ab}$ ).

- (c)  $[2 \xrightarrow{+} 3]$ : Both  $a, b$  are in the same biconnected component, say  $B_i$ , but not in the same rigid component. Let  $a \in V(R_a)$  and  $b \in R_b$ , where  $R_a$  and  $R_b$  are two R-nodes in the SPQR-tree of  $B_i$ . The SPQR-tree of the biconnected component changes after the insertion as follows. All the rigid components on the  $R_a \rightsquigarrow R_b$  SPQR-tree path coalesce into one rigid component (see Figures 3a, 3b). The embedding of the coalesced rigid component is obtained by combining the embeddings of the old triconnected components that are on the  $R_a \rightsquigarrow R_b$  path, with their correct orientation computed from the two-colouring of the separating pairs for the corresponding coherent path. To update the two-colouring of the separating pair vertices, first we discard those old coherent paths and their two-colouring, that are no longer coherent as result of the insertion of  $\{a, b\}$ . While for the subpaths of the old coherent paths that remain coherent we obtain their two-colouring from that of the old path by ignoring colourings of old P-nodes on the  $R_a \rightsquigarrow R_b$  path.
- (d)  $[3 \xrightarrow{+} 3]$ : In this case, both the vertices are in the same 3-connected component. Due to this insertion, connectivity relations do not change. We identify the unique common face in the embedding of the 3-connected component that the two vertices lie on. We split the face into two new faces with the new edge being their common edge, i.e, the face vertex rotation scheme of the old face is split across the new edge. In the vertex rotation scheme of the two vertices, we insert the new edge in an appropriate order.

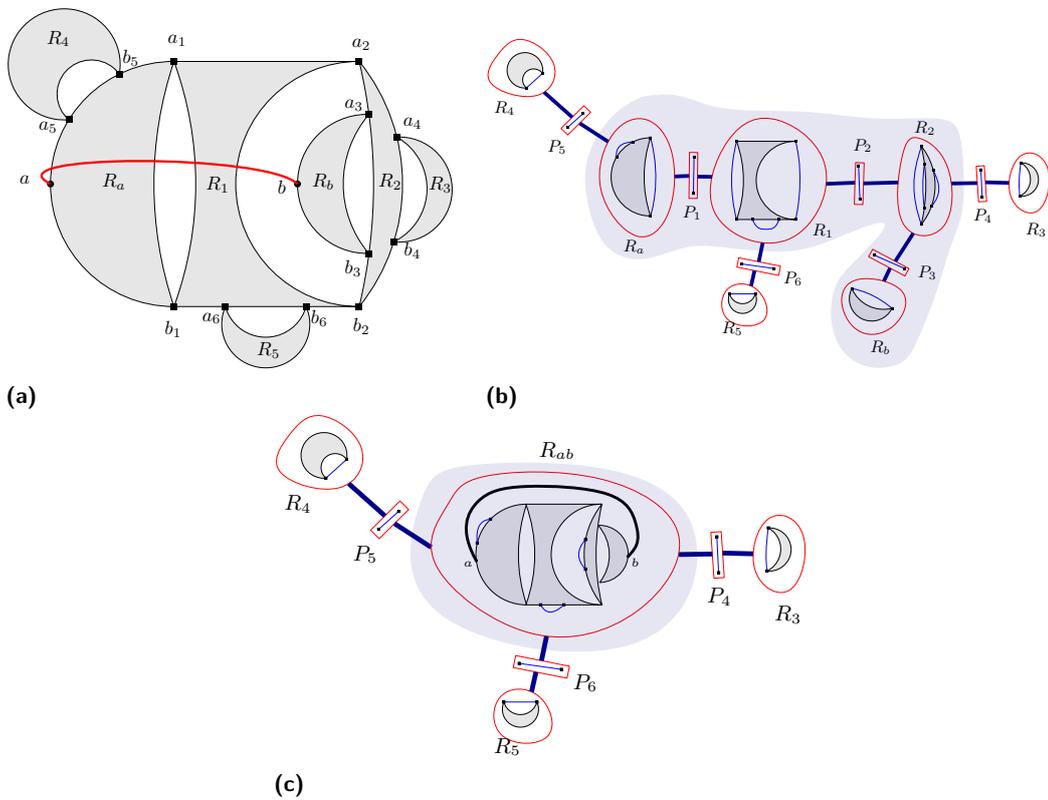
## 5.2 Edge deletion

Find the type of the edge  $\{a, b\}$  that is being deleted. Depending on the type we branch to one of the following options:

- (a)  $[1 \xrightarrow{-} 0]$ : This case occurs when both  $a, b$  are in the same connected component but not in the same biconnected component. The edge is a cut edge and only BC-tree and SPQR-tree relations are affected.
- (b)  $[2 \xrightarrow{-} 1]$ : In this case, both vertices are in the same biconnected component but not in the same 3-connected component before the change. This is the inverse of the insertion operation of an edge of the type  $[1 \xrightarrow{+} 2]$ . The biconnectivity of the old biconnected component changes and it unfurls into a path consisting of multiple biconnected components that are connected via new cut vertices (in Figure 2c if the edge  $\{a, b\}$  is deleted, the block  $B_{ab}$  will unfurl into the highlighted path in Figure 2b). Old virtual edges that are still present in the new biconnected components are deleted. These virtual edge deletions are of type  $[3 \xrightarrow{-} 3]$  or  $[3 \xrightarrow{-} 2]$  which we describe how to handle in the following cases.
- (c)  $[3 \xrightarrow{-} 2]$ : In this case  $a, b$  are in the same rigid component  $R_{ab}$  (see Figure 3c) before the change. But after the deletion of  $\{a, b\}$ ,  $R_{ab}$  decomposes into smaller triconnected components that are unfurled into a path in the SPQR-tree (see Figure 3b). We compute an embedding of the triconnected fragments of  $R_{ab}$  using the embedding of  $R_{ab}$ . From the updated embedding of  $R_{ab}$ , we compute the two-colouring of the separating pair vertices pertaining to the unfurled path on an SPQR-tree. We compose the obtained colouring with the rest of  $B_{ab}$ . The two-colouring of the elongated coherent paths is made consistent by flipping the colouring of the P-nodes of one subpath, if necessary.

(d)  $[3 \bar{\rightarrow} 3]$ : In this case both  $a, b$  are in the same 3-connected component before and after the deletion of  $\{a, b\}$ . Connectivity relations (connectedness, bi/tri-connectivity) remain unchanged. The two faces in the embedding of the 3-connected component that are adjacent to each other via the to-be-deleted edge are merged together to form a new face. From the vertex rotation schemes of the two vertices, the edge is omitted. Pairs of coherent paths that merge due to the merging of two faces have to be consistently two-coloured.

Notice that we do not describe changes of types  $[0 \bar{\rightarrow} 2]$ ,  $[0 \bar{\rightarrow} 3]$ ,  $[1 \bar{\rightarrow} 3]$ ,  $[3 \bar{\rightarrow} 0]$ ,  $[3 \bar{\rightarrow} 1]$ ,  $[2 \bar{\rightarrow} 0]$ , since edge insertion or deletion only changes the number of disjoint paths between any two vertices at most by one and thus these type of edge change are not possible. Also, we omit the changes of type  $[2 \bar{\rightarrow} 2]$  and  $[2 \bar{\rightarrow} 2]$  (related to S-nodes) as they are corner cases that we discuss in the full version of this paper [7, Section 11.4].



■ **Figure 3** A  $[2 \bar{\rightarrow} 3]$  edge  $\{a, b\}$  insertion. (3a) To-be-inserted edge is highlighted in red. (3b) The SPQR-tree of block  $B_i$  before insertion. The path between R-nodes  $R_a$  and  $R_b$  is highlighted. (3c) SPQR-tree of  $B_i$  after insertion. The  $R_a \leftrightarrow R_b$  path has shrunk to a single node  $R_{ab}$ .

### 5.3 Embedding the biconnected components

For each biconnected component, we maintain an embedding in the form of the vertex rotation scheme for its vertices. Note that we do not need to maintain the face-vertex rotation scheme for the biconnected components. The relevant updates for maintaining the biconnected components embedding are edge changes of type  $[1 \bar{\rightarrow} 2]$ ,  $[2 \bar{\rightarrow} 3]$ ,  $[3 \bar{\rightarrow} 3]$ ,  $[3 \bar{\rightarrow} 2]$ , and  $[2 \bar{\rightarrow} 1]$ . For a  $[2 \bar{\rightarrow} 3]$  change, the part of the block that becomes triconnected, after the insertion of the edge, we patch together its vertex rotation scheme with unchanged fragments

of the existing vertex rotation scheme in the affected block. For example, in Figure 3, after the insertion of edge  $\{a, b\}$ , R-nodes  $R_a, R_1, R_2$  and  $R_b$  coalesce together to form  $R_{ab}$  (part of the block that has become triconnected). The embedding of the unchanged fragments of the block, viz.,  $R_3, R_4$  and  $R_5$  is patched together at the appropriate P-nodes with the updated embedding of  $R_{ab}$  to update the embedding of the entire block. In a  $[1 \xrightarrow{+} 2]$  change, multiple biconnected components coalesce together. First, we update the vertex rotation scheme of each block by inserting a required virtual edge and then patch together the vertex rotation schemes at the old cut vertices. For a  $[3 \xrightarrow{-} 2]$  change, we splice in new virtual edges, that arise as a result of the change, in appropriate places in the vertex rotation scheme of the affected block. Details of the updates required in all types of edge changes are described in the full version of this paper [7, Section 12.1].

## 5.4 Embedding the entire graph

Assuming that we have planar embedding (in terms of the vertex rotation scheme) of each biconnected component of the graph we can compute a planar embedding of the whole graph as follows. For the vertices that are not cut vertices, their vertex rotation scheme is the same as their vertex rotation scheme in the embedding of their respective blocks that they belong to. For a cut vertex  $v$ , we join together its vertex rotation schemes from each block  $v$  belongs to. We only need to take care that in the combined vertex rotation scheme of  $v$ , its neighbours in each block appear together and are not interspersed.

We now complete the proof of the main theorem. We provide the complete details in the full version of this paper [7, Section 10, 11 and 12].

► **Lemma 9.** *The extended planar embeddings of all cycle, rigid components and a planar embedding for each biconnected component of the graph can be updated in DynFO under edge changes of type:  $[1 \xrightarrow{+} 2], [2 \xrightarrow{+} 3], [3 \xrightarrow{+} 3], [2 \xrightarrow{+} 2], [2 \xrightarrow{-} 2], [2 \xrightarrow{-} 1], [3 \xrightarrow{-} 2], [3 \xrightarrow{-} 3]$ .*

The above lemma combined with the implementation details provided in the full version [7, Section 12] allows us to prove the main theorem:

► **Theorem 1.** *Given a dynamic graph undergoing insertion and deletion of edges we can maintain a planar embedding of the graph in DynFO (while never allowing insertion of edges that cause the graph to become non-planar).*

**Proof.** If an edge insertion is within a 3-connected component the criterion in Lemma 3 tells us when the resulting component becomes non-planar. Analogously, Lemma 4 informs us when a 2-connected component is non-planar on an edge insertion within it (but across different 3-connected components). Similarly Lemma 6 allows us to determine if an edge added within a connected component but between two different 2-connected components causes non-planarity. An invocation of Lemma 9 shows how to update the planar embedding for biconnected graphs. Moving on to connected graphs, vertex rotation schemes of the cut-vertices is obtained by combining their vertex rotation schemes in each block in any non-interspersed order, using FO primitive *merge vertex rotation schemes* (see [7, Section 9]). Remaining vertices inherit their vertex rotation scheme from the unique block they belong to. This completes the proof. ◀

## 6 Conclusion

We show that planarity testing and embedding is in DynFO where we are able to ensure that an edge is inserted if and only if it does not cause the graph to become non-planar. This is potentially an important step in the direction of solving problems like distance and matching

where only an upper bound of  $\text{DynFO}[\oplus]$  (where  $\text{FO}[\oplus]$  is FO with parity quantifiers) was known in planar graphs. This is because we might be able to improve the known bound making use of planar duality which presupposes a planar embedding. It might also make problems like max flow, graph isomorphism, and counting perfect matchings which are all statically parallelisable when restricted to planar graphs, accessible to a DynFO bound.

---

## References

- 1 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Inf. Comput.*, 189(1):117–134, 2004.
- 2 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within  $\text{NC}^1$ . *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.
- 3 Giuseppe Di Battista and Roberto Tamassia. On-line maintenance of triconnected components with spqr-trees. *Algorithmica*, 15(4):302–318, 1996.
- 4 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
- 5 Samir Datta, Chetan Gupta, Rahul Jain, Anish Mukherjee, Vimal Raj Sharma, and Raghunath Tewari. Dynamic meta-theorems for distance and matching. 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 118:1–118:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 6 Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2014.
- 7 Samir Datta, Asif Khan, and Anish Mukherjee. Dynamic planar embedding is in DynFO, 2023. [arXiv:2307.09473](https://arxiv.org/abs/2307.09473).
- 8 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018.
- 9 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? 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 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 10 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.
- 11 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Log. Methods Comput. Sci.*, 15(2), 2019.
- 12 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, Prague, Czech Republic*, pages 120:1–120:14, 2018.
- 13 Samir Datta and Gautam Prakriya. Planarity testing revisited. In *Theory and Applications of Models of Computation – 8th Annual Conference, TAMC 2011, Tokyo, Japan, May 23-25, 2011. Proceedings*, pages 540–551, 2011.
- 14 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Inc., 5th edition, 2017.
- 15 Hristo N. Djidjev. On drawing a graph convexly in the plane (extended abstract). In Roberto Tamassia and Ioannis G. Tollis, editors, *Graph Drawing*, pages 76–83, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

- 16 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995.
- 17 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.
- 18 Shimon Even and Robert Endre Tarjan. Computing an  $st$ -numbering. *Theor. Comput. Sci.*, 2(3):339–344, 1976.
- 19 F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.
- 20 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Eva Rotenberg. Decremental SPQR-trees for planar graphs. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 46:1–46:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 21 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 167–180, 2020.
- 22 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2378–2397, 2020.
- 23 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
- 24 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.
- 25 Giuseppe F. Italiano, Johannes A. La Poutré, and Monika Rauch. Fully dynamic planarity testing in planar embedded graphs (extended abstract). In *Algorithms – ESA ’93, First Annual European Symposium, Bad Honnef, Germany, September 30 – October 2, 1993, Proceedings*, pages 212–223, 1993.
- 26 Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.
- 27 Saunders Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3(3):460–472, 1937.
- 28 A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. *Theory of Graphs, International Symposium, Rome, July 1966, Rosenstiel, P. edit.*, 1967.
- 29 Jenish C. Mehta. Dynamic complexity of planar 3-connected graph isomorphism. In Edward A. Hirsch, Sergei O. Kuznetsov, Jean-Éric Pin, and Nikolay K. Vereshchagin, editors, *Computer Science – Theory and Applications – 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, volume 8476 of *Lecture Notes in Computer Science*, pages 273–286. Springer, 2014.
- 30 Anish Mukherjee. *Static and Dynamic Complexity of Reachability, Matching and Related Problems*. PhD thesis, CMI, 2019.
- 31 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.
- 32 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 706–715, 1994.
- 33 Vijaya Ramachandran and John H. Reif. Planarity testing in parallel. *J. Comput. Syst. Sci.*, 49(3):517–561, 1994.
- 34 Jeffery R. Westbrook. Fast incremental planarity testing. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 342–353, 1992.
- 35 Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.



# Universality and Forall-Exactness of Cost Register Automata with Few Registers

Laure Daviaud 

School of Computing Sciences, University of East Anglia, Norwich, UK

Andrew Ryzhikov 

Department of Computer Science, University of Oxford, UK

---

## Abstract

The universality problem asks whether a given finite state automaton accepts all the input words. For quantitative models of automata, where input words are mapped to real values, this is naturally extended to ask whether all the words are mapped to values above (or below) a given threshold. This is known to be undecidable for commonly studied examples such as weighted automata over the positive rational (plus-times) or the integer tropical (min-plus) semirings, or equivalently cost register automata (CRAs) over these semirings. In this paper, we prove that when restricted to CRAs with only three registers, the universality problem is still undecidable, even with additional restrictions for the CRAs to be copyless linear with resets.

In contrast, we show that, assuming the unary encoding of updates, the  $\forall$ -exact problem (does the CRA output zero on all the words?) for integer min-plus linear CRAs can be decided in polynomial time if the number of registers is constant. Without the restriction on the number of registers this problem is known to be PSPACE-complete.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Quantitative automata

**Keywords and phrases** cost register automata, universality, forall-exact problem, decidability

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.40

**Funding** *Laure Daviaud*: supported by the EPSRC grant EP/T018313/1.

*Andrew Ryzhikov*: partially supported by the EPSRC grant EP/T018313/1 and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).

## 1 Introduction

Cost register automata (CRAs), introduced by Alur et al. [3], are an extension of finite state automata. Instead of just accepting or rejecting words, they assign each word a value, usually from some semiring. This allows to reason about quantitative properties of systems, such as costs, probabilities, or durations. A CRA is a deterministic finite automaton equipped with a finite set of write-only registers which store values from a semiring, and which are combined using the operations of this semiring. The transitions that a CRA takes thus depend only on the input word, and not on the values of the registers, which makes them different to automata with counters (such as Minsky machines), and allows more of their properties to be decidable.

CRAs are tightly related to weighted automata (WAs), a classical computational model which maps words to values from a fixed semiring. In general, WAs are less expressive than CRAs [3]. However, WAs are equally expressive to linear CRAs, which are CRAs where the updates of the registers are restricted to linear transformations. Hence linear CRAs can be seen as a deterministic model for WAs. Transforming a linear CRA into an equivalent WA and vice versa can be done in polynomial time.



© Laure Daviaud and Andrew Ryzhikov;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 40; pp. 40:1–40:15

Leibniz International Proceedings in Informatics



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

WAs extend automata to a quantitative setting and have been extensively studied since their introduction by Schützenberger in [21], see also surveys [11, 12]. Two widely studied models are WAs over the tropical (or min-plus) semiring and probabilistic WAs, a restricted case of WAs over the semiring of rational numbers with usual addition and multiplication (plus-times semiring). They find their applications in such areas as language and speech processing [19], verification [6], image processing [7], and analysis of on-line algorithms [4] and probabilistic systems [23].

Most applications are algorithmic in their nature, that is, require checking some property of the automaton which models a given system. Often these are classical automata-theoretic properties such as universality, emptiness, boundedness, equivalence, etc., generalised to the quantitative setting. For example, the universality problem for non-deterministic finite state automata asks whether all the input words are accepted. It is PSPACE-complete and is solvable in polynomial time for deterministic finite automata [22]. A natural way to generalise this problem to the quantitative setting is to ask whether all the input words are mapped to a value above (or below, depending on the semiring) a certain threshold. For the min-plus semiring  $(\mathbb{Z} \cup \{+\infty\}, \min, +)$  (are all the values strictly below 0?), and the plus-times semiring  $(\mathbb{Q}, +, \times)$  (are all the values strictly above 1?) this problem is undecidable, see [16, 1] and [20] respectively.

It is thus important to try to find subclasses where this problem becomes decidable. A WA is called linearly ambiguous (respectively, finitely ambiguous) if there is a constant  $C$  such that for every word  $w$  the number of accepting runs labelled by  $w$  is bounded by  $C|w|$  (respectively, by  $C$ ). In both cases (min-plus and plus-times), undecidability is retained even for linearly ambiguous WAs, see [16, 1] and [9] respectively. For finitely ambiguous min-plus WAs universality becomes decidable [24, 13].

Some syntactic restrictions on CRAs allow to introduce subclasses whose expressiveness is incomparable to known classes of WAs. One natural restriction is to bound the number of registers: there exist CRAs with only two registers that compute functions which cannot be computed by a finitely ambiguous WA (Example 2 provides one such CRA). This means that if universality is decidable for CRAs with only two registers, that would allow to decide it for more WAs than it was possible before. We show that for three registers this is not the case, even when restricted further to copyless linear CRAs with resets (see next section for a formal definition), leaving the two-register case as the only remaining option. Undecidability for five registers in the min-plus case follows from a construction in [2], which we use as a starting idea in Subsection 3.1. We note that no characterisation of CRAs with bounded number of registers (or any of its subclasses) in terms of WAs is known, so our results do not follow from any results about WAs.

Informally, a CRA is called copyless if the value of each its registers can only be used once for each transition. In [3], Alur et al. conjectured that universality is decidable for the class of copyless CRAs over the min-plus semiring. In [2], Almagor et al. disproved this conjecture and showed that universality is still undecidable for them. It is natural to ask the same question for the plus-times semiring, but no such results have been known so far. In this paper we show that this conjecture is not true for the plus-times case as well.

CRAs with a bounded number of registers were also studied in the context of register complexity. The register complexity of a function is the minimum number of registers of a CRA that computes it. The problem of computing the register complexity is known to be decidable for unambiguous WA [10], but is open and highly challenging in general. This problem can be seen as a generalisation of the classical determinisation problem, asking if for a given WA there exists an equivalent deterministic one, which amounts to ask whether there exists an equivalent linear CRA with one register [15, 14, 8].

**Our contributions.** In this paper we prove that for CRAs which are linear with resets, copyless and have only 3 registers universality is still undecidable, both for the min-plus and the plus-times semirings. Our approach gives in fact a more general result, encompassing both semirings at the same time, and proves undecidability of universality under some specific conditions on the semiring. This is an additional advantage of our technique, since usually the proofs for the two mentioned semirings are very different.

Another natural decision problem we consider is the  $\forall$ -exact problem, which asks for a given CRA or WA if it outputs zero on all the words. It is known to be PSPACE-complete for WAs (and hence for linear CRAs) over the min-plus semiring [1]. It was also investigated for polynomial automata, which can be seen as a generalisation of WAs over a field [5]. We prove that for this problem bounding the number of registers does help: namely, the  $\forall$ -exact problem is solvable in polynomial time for a linear CRA over the min-plus semiring when the number of its registers is a constant, and the updates of registers are given in the unary encoding.

**Organisation of the paper.** In Section 2, we introduce the model under consideration, namely copyless linear cost-register automata with resets, and the decision problems we study, the universality and the  $\forall$ -exact problems. In Sections 3 and 4, we give the proof of undecidability for the universality problem. To make the content more understandable, we do it in two steps: first we explain the main ideas on a specific sub-problem and on a particular semiring in Section 3, and then extend these ideas to give the general proof in Section 4.

## 2 Cost register automata and decision problems

### 2.1 Cost register automata

Cost register automata (CRAs) are defined in a general way as deterministic finite automata equipped with so-called registers that can store values (numbers, words...) and be combined with operations (addition, multiplication, minimum, discounted sum, concatenation...). In this paper, we consider a quite restrictive class of CRAs. The undecidability results we obtain for this specific class are then applicable to larger classes and CRAs in general.

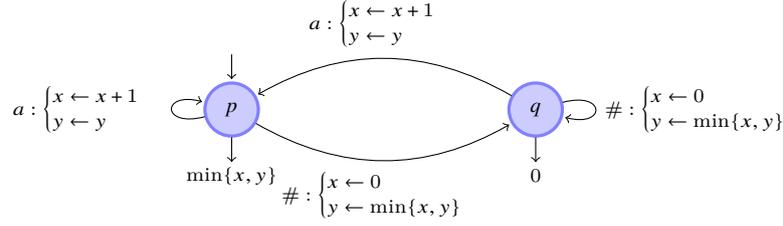
► **Definition 1.** A *linear cost register automaton with resets* with  $k$  registers over a semiring  $(\mathbb{K}, \oplus, \otimes)$  is a deterministic finite automaton  $(Q, \Sigma, \delta, s)$  equipped with registers  $r_1, \dots, r_k$  taking values in  $\mathbb{K}$ , where  $Q$  is a finite set of states and  $s$  the unique initial state,  $\Sigma$  the alphabet and  $\delta$  the transition function. The initial values of the registers are specified by a vector  $\lambda \in \mathbb{K}^k$  and the output function at a state  $q$  is specified as (a linear transformation)  $\oplus_{j=1}^k (m_j^{(q)} \otimes r_j)$  or (a constant)  $m^{(q)}$ , where  $m_1^{(q)}, \dots, m_k^{(q)}, m^{(q)}$  are elements of  $\mathbb{K}$ . Finally, each transition of  $\delta$  is equipped with  $k$  transformations, one for every register, each of one of the two forms:

$$r_i \leftarrow \bigoplus_{j=1}^k m_{i,j} \otimes r_j \text{ (a linear transformation),}$$

$$r_i \leftarrow m_{i,k+1} \text{ (a reset to a constant)}$$

with  $m_{i,j} \in \mathbb{K}$  for all  $1 \leq i \leq k$ ,  $1 \leq j \leq k+1$ .

The semantics of a CRA is defined by means of valuations of the registers. A valuation  $\sigma$  of the registers is a function  $\{r_1, \dots, r_k\} \rightarrow \mathbb{K}$ . A run on a word  $a_1 a_2 \dots a_n$ , where  $a_i \in \Sigma$  for all  $i$ , is a sequence:  $\rho = (q_1, \sigma_1) \xrightarrow{a_1} (q_2, \sigma_2) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_{n+1}, \sigma_{n+1})$  where  $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2}$



■ **Figure 1** An example of CRA on the semiring  $(\mathbb{Z} \cup \{+\infty\}, \min, +)$ .

$\dots \xrightarrow{a_n} q_{n+1}$  is a run in the underlying deterministic finite automaton, and the valuations of the registers  $\sigma_1, \dots, \sigma_{n+1}$  are updated according to the transitions: for all  $1 \leq \ell \leq n$ ,  $\sigma_{\ell+1}(r_i) = \bigoplus_{j=1}^k m_{i,j} \otimes \sigma_\ell(r_j)$  if the transition update is given by a linear transformation and  $\sigma_{\ell+1}(r_i) = m_{i,k+1}$  if it is given by a reset to a constant.

The run is accepting if additionally  $q_1 = s$  and  $\sigma_1(r_i) = \lambda_i$  for all  $i$ , and the output on  $a_1 a_2 \dots a_n$  is given by the output function at state  $q_{n+1}$ , i.e.  $\bigoplus_{j=1}^k m_j^{(q_{n+1})} \otimes \sigma_{n+1}(r_j)$  if this is given by a linear transformation and  $m^{(q_{n+1})}$  if this is given by a constant. Finally, a CRA is called **copyless** if for each transition, every register appears at most once on the right hand-side of the updates: for all  $j$ ,  $m_{i,j}$  is the zero of  $\mathbb{K}$ , except for at most one  $i$ .

All CRAs considered in this paper are from this restrictive class of copyless linear CRAs with resets, so unless specified otherwise by CRAs we mean CRAs from this class. Given a CRA  $\mathcal{A}$ , we will use  $\mathcal{A}$  to denote both the CRA itself and the function  $\Sigma^* \rightarrow \mathbb{K}$  it computes.

► **Example 2.** In Figure 1, we give an example of a CRA over the semiring  $(\mathbb{Z} \cup \{+\infty\}, \min, +)$  and alphabet  $\{a, \#\}$ . It has two states  $p$  (which is initial) and  $q$ , and two registers  $x$  and  $y$ . The output function at state  $p$  (words ending with  $a$  and the empty word) is  $\min\{x, y\}$  and at state  $q$  (words ending with  $\#$ ) is 0. Both registers are initialised with value 0.

If the input word ends with  $\#$  or is empty, this CRA outputs 0. If it ends with  $a$ , the CRA outputs the length of the shortest maximal blocks of consecutive  $a$ 's. Register  $x$  stores the number of  $a$ 's read in the current block, and  $y$  stores the minimum length of the blocks read so far. This CRA is copyless linear with resets.

For any semiring, copyless linear CRAs with resets are at most as expressive as linearly ambiguous WAs, strictly less expressive than polynomially ambiguous WAs, and are incomparable to unambiguous WAs [8]. For the min-plus semiring specifically, they are strictly less expressive than linearly ambiguous WAs [2], incomparable to unambiguous WAs [2] and, at the same time, there exists a copyless CRA (but not linear with resets) with 3 registers which is not equivalent to any polynomially ambiguous WA [17].

## 2.2 Decision problems

We are mainly interested in two classes of CRAs: CRAs over  $(\mathbb{Z} \cup \{+\infty\}, \min, +)$ , called min-plus CRAs, and over  $(\mathbb{Q}, +, \times)$ , called plus-times CRAs in this paper. Our main result is that the universality problem for these two classes, where the number of registers is restricted to 3, is undecidable. We define this problem as follows:

UNIVERSALITY PROBLEM FOR 3-REGISTER CRAs  
 INPUT: A min-plus (respectively, plus-times) CRA  $\mathcal{A}$  with 3 registers.  
 OUTPUT: Yes if and only if for all words  $w$ , we have  $\mathcal{A}(w) < 0$  (respectively,  $\mathcal{A}(w) > 1$ ).

► **Theorem 3.** *The universality problem for 3-register CRAs is undecidable both for min-plus CRAs and for plus-times CRAs.*

To prove this result we give a reduction from the halting problem for Minsky machines with 2 counters, which is known to be undecidable. This leads us to prove a slightly more general result, encompassing both the min-plus and plus-times cases at the same time.

To simulate the behaviour of a Minsky machine with a CRA, we encode the run of the Minsky machine into a specific word and use the CRA to check that a given word corresponds to a correct encoding of the halting run. This boils down to checking some regular properties and verifying that the counters are updated accordingly to the transitions. To explain our approach of this later part, we start by looking at a simpler problem in Section 3: checking that  $n_1 = n_2 = \dots = n_p$  for a word of the shape  $a^{n_1}\#a^{n_2}\#\dots\#a^{n_{p-1}}\#a^{n_p}$ . We explain how to do this with only 3 registers in the min-plus case. In Section 4, we give the full proof of Theorem 3 in a more general setting, allowing us to apply it to both min-plus and plus-times.

On the other hand, we also give a more positive result in the min-plus case: the  $\forall$ -exact problem, as defined below, is known to be PSPACE-complete for linear CRAs with a non-restricted number of registers [1, Theorem 6.13]. We show that it becomes solvable in polynomial time when the number of registers is fixed (and is not a part of the input). For a fixed number of registers, we assume that the size of a CRA is given by its number of states plus the largest absolute value of an integer appearing in an update of a transition or in the output function at a state. Hence, we assume the unary encoding of the numbers in the input. Without the restriction on the number of registers the  $\forall$ -exact problem remains PSPACE-complete in this case [1]. Note that for this results, and for this result only, we consider the class of linear CRAs instead of the class of copyless linear CRAs with resets.

THE  $\forall$ -EXACT PROBLEM FOR  $k$ -REGISTER MIN-PLUS LINEAR CRAS  
 INPUT: A min-plus linear CRA  $\mathcal{A}$  with  $k$  registers.  
 OUTPUT: Yes if and only if for all words  $w$ , we have  $\mathcal{A}(w) = 0$ .

► **Theorem 4.** *For a fixed  $k$ , the  $\forall$ -exact problem is decidable in polynomial time for  $k$ -register min-plus linear CRAs, assuming that the numbers in the transformations are given in the unary encoding.*

This result comes from a variation of a pumping argument, showing that for all words to have value 0, the (useful) values of the registers have to be bounded (below and above) by a constant that is polynomial in the size of the CRA. One can then just keep track of these values. The full proof of this result can be found in the full version of the paper.

**Variants.** The proofs given in this paper can be easily adapted to obtain the undecidability of other variants of the universality problem: for plus-times, whether  $\mathcal{A}(w) > c$  or  $\mathcal{A}(w) < c$ , and for min-plus, whether  $\mathcal{A}(w) < c$  for any constant  $c$ , and the polynomial-time complexity of variants of the  $\forall$ -exact problem: for min-plus, whether  $\mathcal{A}(w) = c$  for any given constant  $c$ . Note that in some cases (for min-plus), a direct translation between these problems is possible and preserves the number of registers. In others (plus-times), the natural translation between these problems would increase the number of registers by 1, but an adaptation of the proofs given in this paper would maintain this number to 3.

**3** Recognising equal-length blocks

To explain our approach, we first look at a simpler problem: given a word

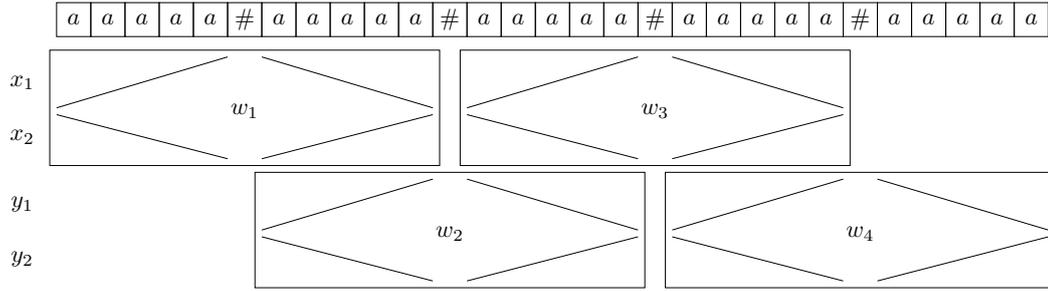
$$w = a^{n_1} \# a^{n_2} \# \dots \# a^{n_{p-1}} \# a^{n_p},$$

check with a min-plus CRA if all  $n_i$  are equal. We assume that the CRA knows that it is going to read the last block of  $a$ 's. For example, that can be done by duplicating the last  $\#$  symbol in the word, but we omit this technical detail for now. More precisely:

► **Proposition 5.** *There exist a min-plus CRA  $\mathcal{A}$  over the alphabet  $\{a, \#\}$  with 3 registers such that  $\mathcal{A}(w) \leq 0$  for all words  $w$ , and  $\mathcal{A}(w) = 0$  if and only if  $w = (a^n \#)^m \# a^n$  for some integers  $n, m$ .*

We describe the solution to this problem in an incremental way, starting from a CRA with 5 registers and then using more and more complex ideas to get to 4 and then finally to only 3 registers. We call a maximal subword of consecutive  $a$ 's a *block*. For  $1 \leq i \leq p - 1$ , define  $w_i = a^{n_i} \# a^{n_{i+1}}$ .

**3.1** Five registers



■ **Figure 2** The computations of two pairs of registers processing  $w_i$ ,  $1 \leq i \leq 4$ . An increasing line means that the register is incremented by 1 for each letter and a decreasing line that the register is decremented by 1 for each letter.

The case of five registers is easy and uses the idea described in [2]. To test if for a word  $w_i = a^{n_i} \# a^{n_{i+1}}$  we have  $n_i = n_{i+1}$ , we use two registers, call them  $x_1$  and  $x_2$ , initialised to zero. While reading the first of two blocks,  $x_1$  is incremented by one, and  $x_2$  is decremented by one. While reading the second block, they do the opposite. Thus, after reading  $w_i$ , the value of  $x_1$  is  $n_i - n_{i+1}$ , and the value of  $x_2$  is  $-n_i + n_{i+1}$ . Observe that  $n_i = n_{i+1}$  if and only if  $\min\{n_i - n_{i+1}, -n_i + n_{i+1}\} = 0$ . Moreover,  $\min\{n_i - n_{i+1}, -n_i + n_{i+1}\}$  is always non-positive. We introduce a new special register  $z$ . After reading  $w_i$ , we set  $z \leftarrow \min\{z, x_1, x_2\}$  and reset  $x_1$  and  $x_2$  to zero. Hence the value of  $z$  is then the minimum of its previous value and  $\min\{n_i - n_{i+1}, -n_i + n_{i+1}\}$ .

After doing that, we have already read the second block of  $w_i$ , so we cannot do the same for  $w_{i+1}$  using the same registers. However, we can cover all words  $w_i$  by using two pairs of registers: one pair  $x_1, x_2$  for  $w_i$  with  $i$  odd, and another pair  $y_1, y_2$  for  $i$  even, because each pair is reset to zero at the end of reading  $w_{i+1}$ , and hence can be used for processing  $w_{i+2}$ . Register  $z$  is shared between these two pairs. Figure 2 illustrates the computations performed by the pairs  $x_1, x_2$  and  $y_1, y_2$ .

The register  $z$  is initialised with zero at the beginning, and changes its value as described above. The output of the CRA is then defined as  $\min\{z, x_1, x_2\}$  if  $i$  is odd, and  $\min\{z, y_1, y_2\}$  if  $i$  is even. This value is the minimum of the values  $\{n_i - n_{i+1}, -n_i + n_{i+1} \mid 1 \leq i \leq p - 1\}$ . It is equal to zero if and only if all  $n_i$  are equal, for  $1 \leq i \leq p$ , otherwise it is strictly negative.

### 3.2 Four registers

The case of four registers is handled similarly, but now we want to get rid of the register  $z$ , and accumulate the non-positive value  $\min\{n_i - n_{i+1}, -n_i + n_{i+1}\}$  in one of the registers that we use for its computation.

Once again, we use two pairs  $x_1, x_2$  and  $y_1, y_2$  of registers to separately process  $w_i$  for odd and even values of  $i$ , and before reading the input we initialise them all with zeros. Assume that  $i$  is odd, and hence we use  $x_1, x_2$  to process the word  $w_i = a^{n_i} \# a^{n_{i+1}}$ . When processing  $w_i$ , the registers  $x_1$  and  $x_2$  perform the same computations as in the case of five registers, thus computing  $n_i - n_{i+1}$  and  $-n_i + n_{i+1}$  respectively. However, after that, instead of sending these values to  $z$  and resetting both registers to zero, we reset to zero only  $x_1$ , and set  $x_2 \leftarrow \min\{x_1, x_2\}$ . Let  $m_i$  be the value of  $x_2$  after processing  $w_i$  if  $i \geq 1$ , and zero otherwise. We can show by induction that  $m_i$  is always non-positive and 0 if and only if  $n_j = n_{j+1}$  for all odd  $j < i$ . Indeed, after processing  $w_i$  the value of  $x_2$  is  $\min\{n_i - n_{i+1}, -n_i + n_{i+1} + m_{i-2}\}$ . By the induction hypothesis, it is always non-positive, and is 0 if and only if  $n_i - n_{i+1} = -n_i + n_{i+1} = 0$  and  $m_{i-2} = 0$ , which concludes the argument. For even  $i$ , we do similarly for  $y_1, y_2$ .

After reading  $w$ , the CRA then outputs the value  $\min\{x_1, x_2, y_1, y_2\}$ . As explained above, this value is zero if and only if for all  $i$  we have  $n_i = n_{i+1}$ , which means that all blocks have the same length. Otherwise this value is strictly negative.

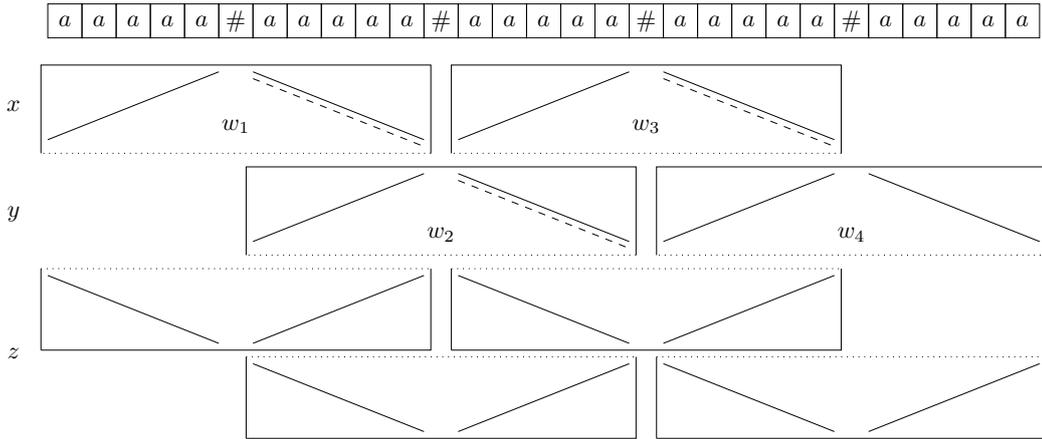
### 3.3 Three registers

**Idea.** For the case of three registers we use the idea described for four registers, but now we only leave the first register of each pair (call these registers  $x$  and  $y$ ), and use the third register (call it  $z$ ) to act simultaneously as the second register from both pairs. Since in the case of four registers the computations performed by two pairs overlap, this will require some adjustments for the behaviour of  $x$  and  $y$  to deal with the overlapping parts.

We use register  $x$  to process  $w_i = a^{n_i} \# a^{n_{i+1}}$  for odd values of  $i$ , and  $y$  for even values of  $i$ . Assume for example that  $i$  is odd. The idea remains the same: we want  $x$  to compute  $n_i - n_{i+1}$ , and  $z$  to compute  $-n_i + n_{i+1}$ . However, if we want  $z$  to perform such computations for all  $w_i$ , then  $z$  is also involved first in processing  $w_{i-1}$  and then in processing  $w_{i+1}$ . The solution is to consider this unwanted change for  $z$  and to make the same change to  $x$ , which then allows to meaningfully compare the values of  $x$  and  $z$  despite the overlaps. Refer to Figure 3 for the symbolic depiction of these computations. The dashed lines illustrate the additional decrements made by  $x$  and  $y$  to adjust for overlap with  $y$  and  $x$  respectively. After processing each  $w_i$ , we set  $z \leftarrow \min\{z, x\}$  if  $i$  is odd, and  $z \leftarrow \min\{z, y\}$  if  $i$  is even.

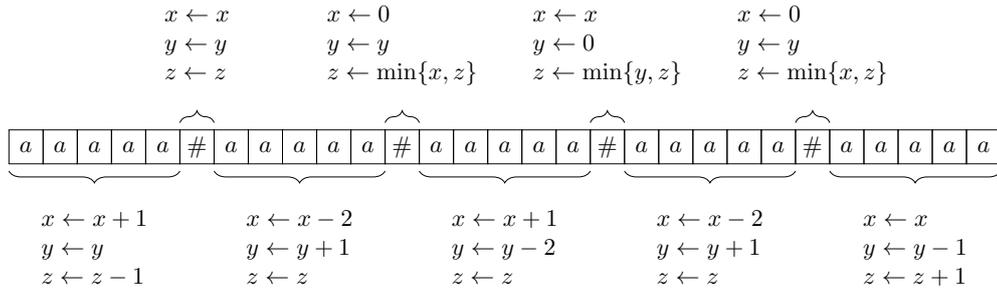
The output of the CRA after reading  $w$  is  $\min\{z, x\}$  if  $i$  is odd, and  $\min\{z, y\}$  if  $i$  is even. Below we show how to construct the CRA implementing this idea.

**Implementation.** As mentioned above, we separate the cases of odd and even  $i$ , and process each  $w_i$  with only one register. At the beginning all registers are initialised with zeros. When processing  $w_i$ ,  $1 \leq i \leq p - 2$ , we compute the value  $n_i - 2n_{i+1}$  by incrementing the corresponding register by one when reading each letter of  $a^{n_i}$ , and decrementing it by two when reading each letter of  $a^{n_{i+1}}$ . After processing  $w_i$ , we set  $z \leftarrow \min\{z, x\}$  if  $i$  is odd, and  $z \leftarrow \min\{z, y\}$  if  $i$  is even. We reset to zero the corresponding used register  $x$  or  $y$ . When processing  $w_{p-1}$  by  $x$  or  $y$ , we compute the value  $n_{p-1} - n_p$  instead.



■ **Figure 3** The computations of the registers processing  $w_i$ ,  $1 \leq i \leq 4$ . An increasing line means that the register is incremented by 1 for each letter and a decreasing line – whether plain or dash – that the register is decremented by 1 for each letter. When both a plain and a dashed line are present it means that the increment/decrement is by 2. The two last lines are for register  $z$ , and hence  $z$  is only incremented/decremented in the first and last block – the increasing and decreasing lines cancel each other but we show them to explain the idea used for the general construction. Note that additionally after reading each block the value of the register  $z$  is changed (not shown in the picture). The dotted sides of the boxes represent that the corresponding boxes from Figure 2 were cut in halves and reassigned to different registers.

The description of the computation is almost done. While reading  $a^{n_1}$ , we decrement  $z$  by one for each letter  $a$ , and while reading  $a^{n_p}$  we increment it by one for each reading of  $a$  (we use here the fact that we know in advance when we are going to read the last block). The output of the CRA is defined as the minimum of the values of the register processing  $w_{p-1}$  (which has the value  $n_{p-1} - n_p$  at the end) and  $z$ . Figure 4 illustrates the computations of thus constructed CRA.



■ **Figure 4** The output value is  $\min\{y, z\}$ .

► **Lemma 6.** *The output of thus constructed CRA is zero if and only if all blocks of  $w$  have the same length, otherwise it is strictly negative.*

**Proof.** After processing  $w_1$ , register  $x$  has value  $n_1 - 2n_2$ , and register  $z$  has value  $-n_1$ . Then  $x$  is reset to 0, and its value is passed to  $z$ , hence the value of  $z$  is  $\min\{-n_1, n_1 - 2n_2\} = -n_2 + \min\{-n_1 + n_2, n_1 - n_2\} = -n_2 + C_1$ , where we define  $C_1 = \min\{n_1 - n_2, -n_1 + n_2\}$ .

For  $2 \leq i \leq p-2$ , define  $C_i = \min\{n_i - n_{i+1}, -n_i + n_{i+1} + C_{i-1}\}$ . Then inductively after reading  $a^{n_{i+1}}$  for  $2 \leq i \leq p-2$  the corresponding register finishes processing  $w_i$  and passes its value to  $z$ , which then has value

$$\min\{n_i - 2n_{i+1}, -n_i + C_{i-1}\} = -n_{i+1} + \min\{n_i - n_{i+1}, -n_i + n_{i+1} + C_{i-1}\} = -n_{i+1} + C_i.$$

Moreover, after reading  $a^{n_p}$  the output of the CRA is  $\min\{n_{p-1} - n_p, -n_{p-1} + n_p + C_{p-2}\}$ , which we accordingly denote by  $C_{p-1}$ . Note that even though the shape of the formula for  $C_{p-1}$  is the same as for  $C_i$  with  $i \leq p-2$ , the way this value is computed by the CRA is different, since while reading  $a^{n_p}$  both  $z$  and the register processing  $w_{p-1}$  behave differently than before.

To show that the constructed CRA satisfies the requirements, we inductively analyse the values  $C_1, \dots, C_{p-1}$ . As noted above,  $C_1 = 0$  if and only if  $n_1 = n_2$ , otherwise  $C_1 < 0$ . Moreover,  $C_i = 0$  if and only if  $n_i = n_{i+1}$  and  $C_{i-1} = 0$ , otherwise  $C_i < 0$ . Hence the output  $C_{p-1} = 0$  if and only if all blocks have the same length, and  $C_{p-1} < 0$  otherwise.  $\blacktriangleleft$

With a similar proof, we can show that the same result applies to plus-times CRA (see end of Section 4 for a general scheme).

## 4 Simulating a Minsky machine with a CRA with 3 registers

In this section, we show how to, given a Minsky machine  $\mathcal{M}$  with two counters, construct a CRA  $\mathcal{A}$  with three registers which simulates  $\mathcal{M}$ . For min-plus, this will mean that  $\mathcal{A}$  outputs 0 on the (unique) word encoding the halting run of  $\mathcal{M}$  if it exists, and outputs a strictly negative value for all other words. For plus-times,  $\mathcal{A}$  outputs 1 on the (unique) word encoding the halting run of  $\mathcal{M}$  if it exists, and outputs a value strictly greater than 1 for all other words. This will prove Theorem 3 as the halting problem for Minsky machine with two counters is undecidable. We will in fact prove a more general result, extending both the min-plus and the plus-times cases. This result is given in Theorem 8.

### 4.1 Minsky machines

Let  $P = \cup_{1 \leq i \leq 2} \{\text{inc}_i, \text{dec}_i, \text{test}_i\}$  be a set of operations (increments, decrements and tests for zero of the  $i$ th counter) on two counters. A Minsky machine  $\mathcal{M}$  with 2 counters is a deterministic finite automaton over the alphabet  $P$ , such that there is a designated *initial* state, and the transitions satisfy the following restrictions: for each state  $q$ , exactly one of the following holds

- $q$  has exactly one outgoing transition, which is then labelled by  $\text{inc}_i$  for some  $i \in \{1, 2\}$ ,
- $q$  has exactly two outgoing transitions, which are then labelled respectively by  $\text{dec}_i, \text{test}_i$  for the same  $i \in \{1, 2\}$ ,
- $q$  has no outgoing transition, in which case it is a unique state called the *halting state*.

Let  $\mathcal{M}$  be a Minsky machine with 2 counters. Consider an alternating sequence

$$\rho = (q_1, \mathbf{v}_1) \xrightarrow{t_1} (q_2, \mathbf{v}_2) \xrightarrow{t_2} \dots \xrightarrow{t_{p-1}} (q_p, \mathbf{v}_p)$$

of pairs  $(q_i, \mathbf{v}_i)$ , where each  $q_i$  is a state and each  $\mathbf{v}_i$  is a pair of non-negative integers, and of operations  $t_i \in P$ . The pairs  $\mathbf{v}_i$  represent the values of the two counters, and we denote by  $\mathbf{v}_i[j]$  its  $j$ th component. Such sequence  $\rho$  is called a *halting run* of  $\mathcal{M}$  if  $q_1 \xrightarrow{t_1} q_2 \xrightarrow{t_2} \dots \xrightarrow{t_{p-1}} q_p$  is a run in the underlying DFA of  $\mathcal{M}$ ,  $q_1$  is the initial state,  $q_p$  is the halting state,  $\mathbf{v}_1 = (0, 0)$ , and for each  $i$ ,  $1 \leq i \leq p-1$ , we have the following (where  $\mathbf{e}_1 = (1, 0)$  and  $\mathbf{e}_2 = (0, 1)$ ):

- if  $t_i = \text{inc}_j$ , then  $\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{e}_j$ ;
- if  $t_i = \text{test}_j$ , then  $\mathbf{v}_i[j] = 0$  and  $\mathbf{v}_{i+1} = \mathbf{v}_i$ ;
- if  $t_i = \text{dec}_j$ , then  $\mathbf{v}_i[j]$  is positive and  $\mathbf{v}_{i+1} = \mathbf{v}_i - \mathbf{e}_j$ .

► **Theorem 7** ([18], Theorem 14.1-1). *The problem whether a given Minsky machine with two counters has a halting run is undecidable.*

## 4.2 Encoding a run of a Minsky machine

To construct a CRA simulating a given Minsky machine with two counters, we first specify how to represent a run of a Minsky machine by a finite word. For now on, fix a Minsky machine  $\mathcal{M}$ .

**Encoding the values of two counters as one number.** If  $\mathbf{v}$  is a vector of two components which are non-negative integer numbers, define  $\nu(\mathbf{v}) = 2^{\nu[1]} \times 3^{\nu[2]}$ . We use  $\nu(\mathbf{v})$  to encode the values of two counters of  $\mathcal{M}$  in a given moment of time. Note that since 2 and 3 are coprime, there is at most one vector  $\nu^{-1}(x)$  for a natural number  $x$ , so this encoding is injective.

**Encoding the runs.** Given a halting run  $\rho = (q_1, \mathbf{v}_1) \xrightarrow{t_1} \dots \xrightarrow{t_{p-1}} (q_p, \mathbf{v}_p)$  of  $\mathcal{M}$ , we construct the word  $W(\rho)$  over the alphabet  $\Sigma = P \cup \{a, \#\}$ , where

$$W(\rho) = t_1 a^{\nu(\mathbf{v}_1)} t_2 a^{\nu(\mathbf{v}_2)} t_3 \dots t_{p-1} a^{\nu(\mathbf{v}_{p-1})} \# a^{\nu(\mathbf{v}_p)}$$

We say that the word  $W(\rho)$  *encodes* the halting run  $\rho$ . Note that each letter  $t_i$  appears before the blocks encoding the values of  $\mathbf{v}_i$  (the argument of  $t_i$ ) and  $\mathbf{v}_{i+1}$  (the result of  $t_i$ ), so that the CRA knows which operation to expect before starting to check that this operation was applied correctly. Note also that the new symbol  $\#$  announces the last block of  $a$ 's. We call the word  $w$  over  $\Sigma$  encoding the halting run of  $\mathcal{M}$  the *run word*.

**Regular tests for run words.** To test if a given word  $w$  over  $\Sigma$  is a run word, we first check if this word has shape

$$t_1 a^{n_1} t_2 a^{n_2} t_3 \dots t_{p-1} a^{n_{p-1}} \# a^{n_p}$$

for some positive integers  $p, n_1, \dots, n_p$ , and for some  $t_1, t_2, \dots, t_{p-1}$  in  $P$ . We then check that there exist states  $q_1, q_2, \dots, q_p$  of  $\mathcal{M}$  such that for each  $1 \leq i \leq p-1$  there is a transition  $q_i \xrightarrow{t_i} q_{i+1}$  in  $\mathcal{M}$ , and also that  $q_1$  is the initial and  $q_p$  is the halting state of  $\mathcal{M}$ .

Next, we check that if  $t_i = \text{test}_\ell$  for  $\ell \in \{1, 2\}$ , we have that the  $\ell$ th component of  $\nu^{-1}(a^{n_i})$  is 0. To do so, we simply check that  $n_i$ , the length of  $a^{n_i}$ , is not divisible by 2 (respectively, 3) if  $\ell = 1$  (respectively,  $\ell = 2$ ). Finally, to test that the values of the counters in the beginning are 0, we simply check that  $n_1 = 1$ .

It is easy to see all these checks are regular, hence they can be performed by the underlying deterministic finite automaton of a CRA, that is, without using any registers. We call a word satisfying these regular properties a *pre-run word*.

What remains to check is that the values of the counters (encoded by  $\nu$ ) change according to the corresponding operations of  $\mathcal{M}$ . This check is the main challenge if we want to perform it by a CRA with a small number of registers. The next section describes how to deal with it.

### 4.3 Handling operations

Let  $\mathcal{M}$  be a Minsky machine, and  $w$  be a pre-run word. We now describe how to construct a CRA  $\mathcal{A}$  with three registers which checks that  $w$  is a run word. The idea of this construction is similar to the idea described in Section 3. The main difference is that now instead of testing that all blocks of  $a$ 's have the same length, we need to test that these lengths are changed according to the corresponding operations.

We give a general construction, that we will be able to apply later on to min-plus and plus-times CRAs and prove the following result, where all the notions will be introduced in due course:

► **Theorem 8.** *Let  $\mathcal{M}$  be a Minsky machine and let  $(\mathbb{K}, \oplus, \otimes, \leq)$  be an ordered semiring with a multiplicative group, such that:*

- *there exists a linear transformation which is a 1-peak (where  $\mathbf{1}$  is the identity element of  $\otimes$ ), and*
- *there exists an element with no finite order.*

*Then one can construct a CRA  $\mathcal{A}$  over  $(\mathbb{K}, \oplus, \otimes)$  such that:*

- *$\mathcal{A}(w) \leq \mathbf{1}$  for all words  $w$ , and*
- *$\mathcal{A}(w) = \mathbf{1}$  if and only if  $w$  encodes the halting run of  $\mathcal{M}$ .*

We consider a semiring  $(\mathbb{K}, \oplus, \otimes)$ . It is said to have a multiplicative group if  $(\mathbb{K} - \{\mathbf{0}\}, \otimes)$  is a group, where  $\mathbf{0}$  is the identity element of  $\oplus$ . We denote by  $\mathbf{1}$  the identity element of this group. We also denote, for an element  $d$  of the group and some positive integer  $\ell$ , by  $d^\ell$  the product of  $d$  by itself  $\ell$  times, and by  $d^{-\ell}$  its inverse. By convention  $d^0 = \mathbf{1}$ . We also fix a linear transformation  $\varphi : \mathbb{K}^2 \rightarrow \mathbb{K}$ , i.e.  $\varphi(x, y) = (c \otimes x) \oplus (d \otimes y)$  for some  $c, d$  in  $\mathbb{K}$ , and we fix an element  $\alpha$  of  $\mathbb{K}$ .

We will now construct a CRA over the semiring  $(\mathbb{K}, \oplus, \otimes)$  using  $\alpha$  and  $\varphi$ .

Let  $w = t_1 a^{n_1} t_2 a^{n_2} t_3 \dots t_{p-1} a^{n_{p-1}} \# a^{n_p}$  be a pre-run input word. For convenience of the presentation, we denote  $t_p = \#$ . As before, we call a maximal continuous subword of  $w$  consisting of  $a$ 's a *block*. Denote  $w_i = t_i a^{n_i} t_{i+1} a^{n_{i+1}}$ . Call  $x, y, z$  the three registers of  $\mathcal{A}$ . We process  $w_i$  with  $x$  for odd values of  $i$ , and with  $y$  for even values of  $i$ , and send the results to  $z$ .

First we describe what  $\mathcal{A}$  does when reading  $a^{n_i}$  for  $1 < i < p$ . The word  $a^{n_i}$  is the second block in  $w_{i-1}$  and the first block in  $w_i$ . Assume without loss of generality that  $i$  is odd (otherwise switch  $x$  and  $y$  in the further description). Then  $w_i$  is processed by  $x$ , and  $w_{i-1}$  is processed by  $y$ . The computations performed by  $x$  depend only on  $t_i$ : for each occurrence of  $a$ , we set

- $x \leftarrow x \otimes \alpha$  if  $t_i \in \{\text{test}_\ell, \text{dec}_\ell\}$  for  $\ell \in \{1, 2\}$ ,
- $x \leftarrow x \otimes \alpha^2$  if  $t_i = \text{inc}_1$ ,
- $x \leftarrow x \otimes \alpha^3$  if  $t_i = \text{inc}_2$ .

The computations performed by  $y$  depend on both  $t_{i-1}$  and  $t_i$ . Let  $c$  be the value (1, 2 or 3) used above in the transformation  $x \leftarrow x \otimes \alpha^c$  depending on  $t_i$ . Then for each occurrence of  $a$  we set:

- $y \leftarrow y \otimes \alpha^{-1} \otimes \alpha^{-c}$  if  $t_{i-1} \in \{\text{test}_\ell, \text{inc}_\ell\}$  for  $\ell \in \{1, 2\}$ ,
- $y \leftarrow y \otimes \alpha^{-2} \otimes \alpha^{-c}$  if  $t_{i-1} = \text{dec}_1$ ,
- $y \leftarrow y \otimes \alpha^{-3} \otimes \alpha^{-c}$  if  $t_{i-1} = \text{dec}_2$ .

Furthermore, if, for each letter of  $a^{n_i}$ , register  $x$  is multiplied by  $\alpha^c$  and  $y$  is multiplied by  $\alpha^{-d} \otimes \alpha^{-c}$ , then we set  $z \leftarrow z \otimes \alpha^{-c} \otimes \alpha^d$  for each letter of  $a^{n_i}$ . After reading  $t_{i+2}$ , the next letter after  $a^{n_i}$ , we set  $z \leftarrow \varphi(y, z)$ , and reset  $y$  to  $\mathbf{1}$ .

It remains to describe the transformations of the registers for  $a^{n_1}$  and  $a^{n_p}$ . For each letter of  $a^{n_1}$ ,  $x$  performs the same update as described above, and  $z$  is updated by the inverse of that value. Assume without loss of generality that  $p - 1$  is even (otherwise use  $x$  instead of  $y$  in the further description). Then  $w_{p-1}$  is processed by  $y$ . For each letter in  $a^{n_p}$  we set

- $y \leftarrow y \otimes \alpha^{-1}, z \leftarrow z \otimes \alpha$  if  $t_{p-1} \in \{\text{test}_\ell, \text{inc}_\ell\}$  for  $\ell \in \{1, 2\}$ ,
- $y \leftarrow y \otimes \alpha^{-2}, z \leftarrow z \otimes \alpha^2$  if  $t_{p-1} = \text{dec}_1$ ,
- $y \leftarrow y \otimes \alpha^{-3}, z \leftarrow z \otimes \alpha^3$  if  $t_{p-1} = \text{dec}_2$ .

Finally, we set the output of  $\mathcal{A}$  to be  $\varphi(y, z)$ . Note that in this construction, the CRA is linear with resets and copyless.

► **Example 9.** Consider the following halting run:

$$\rho = (q_1, (0, 0)) \xrightarrow{\text{inc}_2} (q_1, (0, 1)) \xrightarrow{\text{inc}_1} (q_1, (1, 1)) \xrightarrow{\text{dec}_1} (q_1, (0, 1)) \xrightarrow{\text{test}_1} (q_1, (0, 1))$$

We abbreviate by  $t_1, t_2, t_3, t_4$  the operations along this run and by  $t_5$  the  $\#$  symbol indicating that we are going to read the last block. Note again that in  $w$  the operation is written before its argument, hence for example  $t_1 = \text{inc}_2$  is applied to the first block, and its result is the second block. The computations of  $\mathcal{A}$  for the word encoding  $\rho$  are shown in Figure 5. The CRA outputs  $\varphi(y, z)$  in this example.

$x \leftarrow x$	$x \leftarrow \mathbf{1}$	$x \leftarrow x$	$x \leftarrow \mathbf{1}$
$y \leftarrow y$	$y \leftarrow y$	$y \leftarrow \mathbf{1}$	$y \leftarrow y$
$z \leftarrow z$	$z \leftarrow \varphi(x, z)$	$z \leftarrow \varphi(y, z)$	$z \leftarrow \varphi(x, z)$

$t_1$	$a$	$t_2$	$a$	$a$	$a$	$t_3$	$a$	$a$	$a$	$a$	$a$	$t_4$	$a$	$a$	$a$	$t_5$	$a$	$a$	$a$
-------	-----	-------	-----	-----	-----	-------	-----	-----	-----	-----	-----	-------	-----	-----	-----	-------	-----	-----	-----

$x \leftarrow x \otimes \alpha^3$	$x \leftarrow x \otimes \alpha^{-3}$	$x \leftarrow x \otimes \alpha$	$x \leftarrow x \otimes \alpha^{-3}$	$x \leftarrow x$
$y \leftarrow y$	$y \leftarrow y \otimes \alpha^2$	$y \leftarrow y \otimes \alpha^{-2}$	$y \leftarrow y \otimes \alpha$	$y \leftarrow y \otimes \alpha^{-1}$
$z \leftarrow z \otimes \alpha^{-3}$	$z \leftarrow z \otimes \alpha^{-1}$	$z \leftarrow z$	$z \leftarrow z \otimes \alpha$	$z \leftarrow z \otimes \alpha$

■ **Figure 5** The computations of  $\mathcal{A}$  on the word encoding the halting run  $\rho$ .

Suppose now that  $\mathbb{K}$  is equipped with a linear order  $\leq$ , and denote by  $<$  the associated strict order (we call such  $\mathbb{K}$  an *ordered semiring*). We say that  $\alpha$  has no finite order if  $\alpha^\ell = \mathbf{1}$  implies  $\ell = 0$  and that  $\varphi$  is a *1-peak* if, for all  $d, e$  in  $\mathbb{K}$  such that  $d \otimes e \leq \mathbf{1}$ , we have  $\varphi(d, e) = \mathbf{1}$  if and only if  $d = e = \mathbf{1}$ , and otherwise  $\varphi(d, e) < \mathbf{1}$ .

► **Lemma 10.** *Suppose that  $\varphi$  is a 1-peak and that  $\alpha$  has no finite order. Then, for a pre-run word  $w$ , we have  $\mathcal{A}(w) = \mathbf{1}$  if and only if  $w$  is a run word, otherwise  $\mathcal{A}(w) < \mathbf{1}$ .*

**Proof.** We adapt the proof of Lemma 6. Define  $C_0 = \mathbf{1}$  and for  $1 \leq i \leq p - 1$  define

$$C_i = \begin{cases} \{\varphi(\alpha^{n_i - n_{i+1}}, \alpha^{-n_i + n_{i+1}} \otimes C_{i-1})\} & \text{if } t_i = \text{test}_\ell \text{ for } \ell = 1, 2 \\ \{\varphi(\alpha^{2n_i - n_{i+1}}, \alpha^{-2n_i + n_{i+1}} \otimes C_{i-1})\} & \text{if } t_i = \text{inc}_1 \\ \{\varphi(\alpha^{3n_i - n_{i+1}}, \alpha^{-3n_i + n_{i+1}} \otimes C_{i-1})\} & \text{if } t_i = \text{inc}_2 \\ \{\varphi(\alpha^{n_i - 2n_{i+1}}, \alpha^{-n_i + 2n_{i+1}} \otimes C_{i-1})\} & \text{if } t_i = \text{dec}_1 \\ \{\varphi(\alpha^{n_i - 3n_{i+1}}, \alpha^{-n_i + 3n_{i+1}} \otimes C_{i-1})\} & \text{if } t_i = \text{dec}_2 \end{cases}$$

Observe that by construction after a register ( $x$  or  $y$ ) processes  $w_i$ ,  $1 \leq i \leq p - 2$ , and passes its value to  $z$ , the value of  $z$  is

$$\begin{cases} \alpha^{-n_{i+1}} \otimes C_i & \text{if } t_{i+1} = \text{test}_\ell \text{ or } t_{i+1} = \text{inc}_\ell \text{ for } \ell \in \{1, 2\} \\ \alpha^{-2n_{i+1}} \otimes C_i & \text{if } t_{i+1} = \text{dec}_1 \\ \alpha^{-3n_{i+1}} \otimes C_i & \text{if } t_{i+1} = \text{dec}_2 \end{cases}$$

Moreover, the output of  $\mathcal{A}$  equals  $C_{p-1}$ .

Observe that using the definition of  $C_i$  and by induction, one can prove that  $C_i = \mathbf{1}$  if and only if  $t_i$  is performed correctly (that is, the result of  $t_i$  on the pair  $v^{-1}(a^{n_i})$  is  $v^{-1}(a^{n_{i+1}})$ ) and  $C_{i-1} = \mathbf{1}$ , otherwise  $C_i < \mathbf{1}$ , since  $\varphi$  is  $\mathbf{1}$ -peak and  $\alpha$  has no finite order. Hence the output  $C_{p-1}$  equals  $\mathbf{1}$  if and only if the input is a run word, otherwise it is  $< \mathbf{1}$ . ◀

By combining this proposition with the checks for pre-run words described in Section 4.2, we get a CRA which outputs  $\mathbf{1}$  if and only if the input encodes a halting run of  $\mathcal{M}$ . Indeed, if a word is not a pre-run word (which is checked by the underlying DFA of the CRA), the run labelled by it will end in a state which outputs a constant value  $f < \mathbf{1}$ . All runs ending in other states are thus labelled by pre-run words, and for each such word the computations described in this section output the desired value depending on whether this word encodes a halting run or not. This means that all runs in the CRA are considered, concluding the proof of Theorem 8.

**Min-plus and plus-times cases.** To finish the proof of Theorem 3, it is enough to instantiate  $(\mathbb{K}, \oplus, \otimes, \leq)$ ,  $\alpha$  and  $\varphi$  to suitable elements. For min-plus, we take  $(\mathbb{K}, \oplus, \otimes, \leq) = (\mathbb{Z} \cup \{+\infty\}, \min, +, \leq)$ ,  $\alpha = 1$  and  $\varphi = \min$ . For plus-times, we take  $(\mathbb{K}, \oplus, \otimes, \leq) = (\mathbb{Q}_+, +, \times, \geq)$ ,  $\alpha = 2$  and  $\varphi(c, d) = 2^{-1}(c + d)$ , where  $\mathbb{Q}_+$  is the set of positive rational numbers. It is easy to check that in both cases,  $\varphi$  is  $\mathbf{1}$ -peak and  $\alpha$  has no finite order.

## 5 Conclusions

In this paper, we prove the undecidability of the universality problem for models of CRAs where the number of registers is limited to 3. Our main result holds for min-plus and plus-times CRAs, but we give a slightly more general construction and it would be interesting to see if our techniques can be applied to other cases, and in particular to see its link to infinitary groups that have already been studied in conjunction to the register complexity [10].

The main open question that remains is whether this is still true when considering CRAs with only 2 registers. Our proof cannot be adapted easily to the 2-register case. One approach is to understand whether with only 2 registers, one can recognise the language with equal length blocks as defined in Section 3. Even this is difficult.

Finally, we proved that the  $\forall$ -exact problem is solvable in polynomial time when the number of registers of a min-plus linear CRA is fixed. The same question can be asked for the boundedness problem over the  $(\mathbb{N} \cup \{+\infty\}, \min, +)$  semiring, which is known to be PSPACE-complete for WAs (and hence linear CRAs) [1].

---

## References

- 1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? *Information and Computation*, 282:104651, 2022. doi:10.1016/j.ic.2020.104651.
- 2 Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak cost register automata are still powerful. *International Journal of Foundations of Computer Science*, 31(6):689–709, 2020. doi:10.1142/S0129054120410026.

- 3 Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*, pages 13–22, 2013. doi:10.1109/LICS.2013.65.
- 4 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2):28:1–28:36, 2010. doi:10.1145/1721837.1721844.
- 5 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005101.
- 6 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.
- 7 Karel Culík and Jarkko Kari. Digital images and formal languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 599–616. Springer, 1997. doi:10.1007/978-3-642-59126-6\_10.
- 8 Laure Daviaud. Register complexity and determinisation of max-plus automata. *ACM SIGLOG News*, 7(2):4–14, 2020. doi:10.1145/3397619.3397621.
- 9 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When are emptiness and containment decidable for probabilistic automata? *Journal of Computer and System Sciences*, 119:78–96, 2021. doi:10.1016/j.jcss.2021.01.006.
- 10 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS 2016)*, pages 857–866, 2016. doi:10.1145/2933575.2934549.
- 11 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Berlin, Heidelberg, 1st edition, 2009. doi:10.1007/978-3-642-01492-5.
- 12 Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/Automata-1/4.
- 13 Kosaburo Hashiguchi, Kenichi Ishiguro, and Shuji Jimbo. Decisability of the equivalence problem for finitely ambiguous automata. *International Journal of Algebra and Computation*, 12(03):445–461, 2002. doi:10.1142/S0218196702000845.
- 14 Daniel Kirsten and Sylvain Lombardy. Deciding Unambiguity and Sequentiality of Polynomially Ambiguous Min-Plus Automata. In *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3 of *LIPICs*, pages 589–600, 2009. doi:10.4230/LIPICs.STACS.2009.1850.
- 15 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.
- 16 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–426, 1994. doi:10.1142/S0218196794000063.
- 17 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, 2019. doi:10.1016/j.jcss.2018.07.002.
- 18 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, USA, 1967.
- 19 Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997. URL: <https://aclanthology.org/J97-2003>.
- 20 Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 1971.
- 21 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

- 22 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9, 1973. doi:10.1145/800125.804029.
- 23 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science (FOCS 1985)*, pages 327–338, 1985. doi:10.1109/SFCS.1985.12.
- 24 Andreas Weber. Finite-valued distance automata. *Theoretical Computer Science*, 134(1):225–251, 1994. doi:10.1016/0304-3975(94)90287-9.



# Relaxed Core Stability for Hedonic Games with Size-Dependent Utilities

Tom Demeulemeester    
KU Leuven, Belgium

Jannik Peters    
TU Berlin, Germany

---

## Abstract

We study relationships between different relaxed notions of core stability in hedonic games. In particular, we study (i)  $q$ -size core stable outcomes in which no deviating coalition of size at most  $q$  exists and (ii)  $k$ -improvement core stable outcomes in which no coalition can improve by a factor of more than  $k$ . For a large class of hedonic games, including fractional and additively separable hedonic games, we derive upper bounds on the maximum factor by which a coalition of a certain size can improve in a  $q$ -size core stable outcome. We further provide asymptotically tight lower bounds for a large class of hedonic games. Finally, our bounds allow us to confirm two conjectures by Fanelli et al. [20][IJCAI'21] for symmetric fractional hedonic games (S-FHGs): (i) every  $q$ -size core stable outcome in an S-FHG is also  $\frac{q}{q-1}$ -improvement core stable and (ii) the price of anarchy of  $q$ -size stability in S-FHGs is precisely  $\frac{2q}{q-1}$ .

**2012 ACM Subject Classification** Computing methodologies → Multi-agent systems; Theory of computation → Algorithmic game theory

**Keywords and phrases** hedonic games, core stability, algorithmic game theory, computational social choice

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.41

**Supplementary Material** *Software*: <https://github.com/DemeulemeesterT/Relaxations-Core-Stability-Alpha-Hedonic-Games>  
archived at `swh:1:dir:45f549c91743070a02a2f53d19ed280716a335f6`

**Funding** *Tom Demeulemeester*: Research Foundation – Flanders (FWO) under the PhD fellowship 11J8721N.

*Jannik Peters*: Deutsche Forschungsgemeinschaft (DFG) under the grant BR 4744/2-1 and the Graduiertenkolleg “Facets of Complexity” (GRK 2434).

## 1 Introduction

Coalition formation is one of the core topics of multiagent systems and algorithmic game theory. Hedonic games ([19]) constitute the most popular subcase of coalition formation. In a hedonic game, the goal is to divide a set of agents into disjoint coalitions, respecting the preferences of the agents. Over the years, multiple different ways of representing the agents’ preferences and multiple different solution concepts emerged. Among the strongest solution concepts is core stability ([9]): a coalition structure is core stable, if no subset of agents could together form a new coalition in which they are all better off than in the original coalition structure. While being a seemingly natural concept, it has been shown that even for very simple preference structures, core stable outcomes may not exist ([2, 3]). Further, in these structures, it is also often computationally intractable to decide whether a core stable



© Tom Demeulemeester and Jannik Peters;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics



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

outcome exists.<sup>1</sup> These results led Fanelli et al. [20] to introduce two natural weakenings of core stability: (i)  $q$ -size (core) stability, which requires that no blocking coalitions of size at most  $q$  exist, (ii)  $k$ -improvement (core) stability, which requires that no blocking coalition exists in which every agent improves by a factor of more than  $k$ . For the example of fractional hedonic games ([2]), [20] showed that a 2-size stable outcome and a 2-improvement stable outcome always exist. Further, they also studied the relationship between these two notions and were able to show that a 2-size stable outcome is indeed also always 2-improvement stable.

In this paper, we contribute to this literature in two ways. First, we propose a new class of hedonic games, called  $\alpha$ -hedonic games, in which the utility an agent receives from being in a coalition of size  $m$  is equal to the sum of the cardinal utilities it ascribes to the other agents in that coalition, multiplied by a factor  $\alpha_m$  which depends on the size of that coalition. Several well-studied classes of hedonic games, such as fractional hedonic games [2], modified fractional hedonic games [27], and additively separable hedonic games [9] are a special case of  $\alpha$ -hedonic games.

Second, we further study the two weakenings of core stability that were introduced by Fanelli et al. [20]. Our main result quantifies, for any  $\alpha$ -hedonic game and for any  $q$ -size stable outcome, the maximum factor with which the agents can improve their utility by forming a blocking coalition of size  $m \geq q + 1$ . As a corollary, this allows us to prove two conjectures by Fanelli et al. [20]: (i) every  $q$ -size stable outcome is  $\frac{q}{q-1}$ -improvement stable for fractional hedonic games and (ii) the  $q$ -size core price of anarchy (i.e., the worst-case approximation to the social welfare of any  $q$ -size core stable outcome) is exactly  $\frac{2q}{q-1}$  for fractional hedonic games.

## Related Work

Since its inception hedonic games have been a widely studied topic in algorithmic game theory, with several works studying axiomatic or computational properties of hedonic games. For an overview on earlier developments, we refer the reader to the book chapter by Aziz and Savani [1]. In recent years, several new models and optimality notions for hedonic games were introduced and analyzed. Among the most popular of these notions are the aforementioned fractional hedonic games, introduced by Aziz et al. [2] and studied in various forms by, e.g., Bilò et al. [5], Aziz et al. [4], or Carosi et al. [15]. Fractional hedonic games are also related to the model of hedonic diversity games [13, 8, 21] in which agents possess types and derive utility based on the fraction of agents of their own type in their coalition.

The paper closest to ours is the work by Fanelli et al. [20], who introduced the aforementioned notions of  $q$ -size and  $k$ -improvement core stability for fractional hedonic games. Alternative simplifications of core stability were introduced by Carosi et al. [15], who studied a local variant of core stability for simple fractional hedonic games, i.e., hedonic games in which all utility values are either 0 or 1. In their local variant of core stability, the agents deviating are required to form a clique. For this weakened notion, they show that core stable outcomes always exist and can be computed via improving response dynamics.

Finally, some very recent works on hedonic games include [7, 12, 11] who study various aspects of dynamics, i.e., decentralized processes in which agents perform beneficial changes until a stable outcome is reached, the study of coalition formation with (almost) fixed

---

<sup>1</sup> For some preference structures, it can even be hard to find a coalition structure in the core, even if it is guaranteed to exist, see Bullinger and Kober [14].

coalition sizes [18, 6, 25, 24], or studies of the complexity of various hedonic games variants [17, 16, 22, 10]. Another interesting direction are the models of altruistic [23] and loyal [14] hedonic games, in which the utilities of agents do not only depend on their own utility, but also on the utility of their friends/agents they are loyal to.

## 2 Preliminaries

For any  $n \in \mathbb{N}^+$  and  $\alpha: [n] \rightarrow \mathbb{R}^+$  an  $\alpha$ -hedonic game ( $\alpha$ HG) consists of a set of agents  $A = \{a_1, \dots, a_n\}$  with a utility function  $u: A \times A \rightarrow \mathbb{R}$ . We restrict ourselves to symmetric  $\alpha$ -hedonic games (S- $\alpha$ HGs) in this paper, and require that  $u(i, j) = u(j, i)$  for all  $i, j \in A$ . A coalition is a subset of  $A$  and a coalition structure is a partition of  $A$  into coalitions. The utility of an agent  $i$  in a coalition  $C$  is  $u_i(C) := \sum_{j \in C} \alpha(|C|) \cdot u(i, j)$ . We assume that  $u(i, i) = 0$ . For a coalition structure  $\mathcal{C}$  the utility  $u_i(\mathcal{C})$  of the coalition structure for agent  $i$  is the utility of the coalition agent  $i$  belongs to. To simplify notation, for agents  $a_i$  and  $a_j$  and  $C \subseteq A$  we also write  $u_i(a_j) := u(a_i, a_j)$  and  $u_i(C) := u_{a_i}(C)$ .

The class of  $\alpha$ -hedonic games generalizes multiple previously studied hedonic game classes, e.g.,

- Symmetric additively separable hedonic games (S-ASHGs) with  $\alpha(m) = 1$  for any  $m \in \mathbb{N}$ .
- Symmetric fractional hedonic games (S-FHG) with  $\alpha(m) = \frac{1}{m}$  for any  $m \in \mathbb{N}$ .
- Symmetric modified fractional hedonic games (S-MFHGs) with  $\alpha(m) = \frac{1}{m-1}$  for any  $m \in \mathbb{N}^+$  and  $\alpha(1) = 0$ .

A given coalition structure  $\mathcal{C}$  is

- core stable if for any coalition  $C$  it holds that  $u_i(C) \leq u_i(\mathcal{C})$  for at least one  $i \in C$ ;
- $q$ -size core stable if for any coalition  $C$  with  $|C| \leq q$  it holds that  $u_i(C) \leq u_i(\mathcal{C})$  for at least one  $i \in C$ ;
- $k$ -improvement core stable if for any coalition  $C$  it holds that  $u_i(C) \leq k u_i(\mathcal{C})$  for at least one  $i \in C$ ;
- $(q, k)$ -core stable if for any coalition  $C$  with  $|C| = q$  it holds that  $u_i(C) \leq k u_i(\mathcal{C})$  for at least one  $i \in C$ .

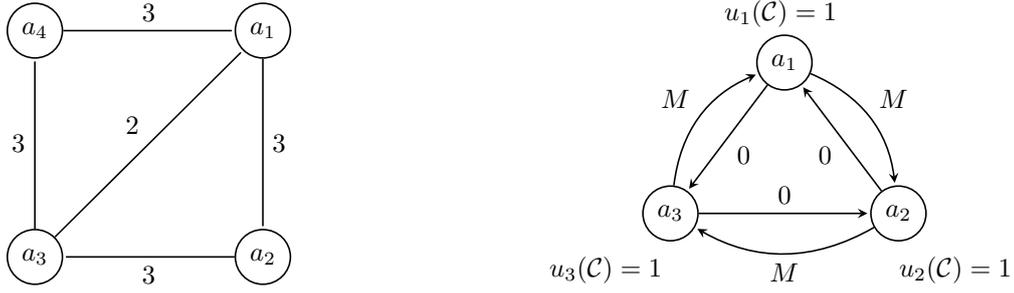
If there is a coalition witnessing a violation to one of these criteria, for instance a coalition  $C$  with  $u_i(C) > u_i(\mathcal{C})$  for all  $i \in C$ , we say that  $C$  is a blocking coalition. In further parts of the paper, we shorten  $\alpha(m)$  to  $\alpha_m$  to increase readability. Moreover, denote by  $\mathbb{1}: \mathbb{N} \rightarrow \{0, 1\}$  the indicator function such that for any  $i \in \mathbb{N}$

$$\mathbb{1}(i) = \begin{cases} 0 & \text{if } i = 0, \\ 1 & \text{else.} \end{cases}$$

Lastly, every S- $\alpha$ HG can be represented by a graph  $G(A, E, w)$ , where  $A$  represents the set of agents, and  $E$  contains an undirected edge  $\{i, j\}$  between agents  $i$  and  $j$  with weight  $w_{ij} = u(i, j) = u(j, i)$  if  $u(i, j) > 0$ . Alternatively, given a coalition  $C \subseteq A$ , we denote the subgraph of  $G(A, E, w)$  that is induced by only considering the agents in  $C$  by  $G(C)$ .

Before turning to our result, we present two simple examples, with the second example motivating why we exclusively focus on symmetric instances.

► **Example 1.** First, consider the hedonic game induced by the graph on the left of Figure 1, with utilities as indicated by the edges and omitted edges indicating a utility of 0. Here, consider the coalition structure  $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ . In an S-ASHG, the utility of every agent would be 3, and the coalition structure would be 2-size core stable, but not 3-size core stable, since, for instance,  $\{a_1, a_2, a_3\}$  would block. Further, the coalition structure is  $(3, \frac{5}{3})$ - and



■ **Figure 1** Example of a symmetric hedonic game on the left and of a blocking coalition for an asymmetric game on the left, for which the improvement ratio is unbounded.

(4, 2)-core stable and thus also 2-improvement core stable. In an S-FHG, on the other hand, the utility of every agent would be  $\frac{3}{2}$  and the coalition  $\{a_1, a_2, a_3\}$  would still block. The coalition consisting of all agents, however, would no longer be blocking, since the utility of agent  $a_2$  would be  $\frac{6}{4} = \frac{3}{2}$ , which was their utility in the original coalition structure. Finally, in an S-MFHG, the utility of every agent would be 3 and the coalition structure would be core stable. Even the coalition  $\{a_1, a_2, a_3\}$  would no longer block, since the utility of agent  $a_1$  would be  $\frac{5}{2} < 3$ .

Secondly, to motivate the choice of symmetric hedonic games, consider the (asymmetric) hedonic game depicted on the right of Figure 1 with all three agents originally being in a coalition structure  $\mathcal{C}$  in which they experience utility 1. This coalition structure would be 2-size core stable. However, there is no upper bound on the improvement ratio for the coalition consisting of all three agents, as  $M$  goes to infinity. We note that this behaviour can be observed independently of the considered  $\alpha$  function.

## 2.1 Our results

Fanelli et al. [20] conjectured that for fractional hedonic games, every  $q$ -size core stable coalition structure is also  $\frac{q}{q-1}$ -improvement core stable. We refine this conjecture and show that every  $q$ -size core stable coalition structure is

$$\left( m, 1 + \frac{\lfloor \frac{1}{q-1}(m-2) \rfloor}{m} \right) \text{-core stable} \quad (1)$$

for any  $m \geq q + 1$ . As  $1 + \frac{\lfloor \frac{1}{q-1}(m-2) \rfloor}{m} \leq \frac{q}{q-1}$  for any  $m$  this implies the conjecture of Fanelli et al. [20]. Further, this result together with the results of Fanelli et al. [20] also allows us to confirm their second conjecture that the *price of anarchy* of  $q$ -size stability is exactly  $\frac{2q}{q-1}$ .

To gain a better intuition of this quite unhandy term, we refer the reader to Table 1 and Figure 2.

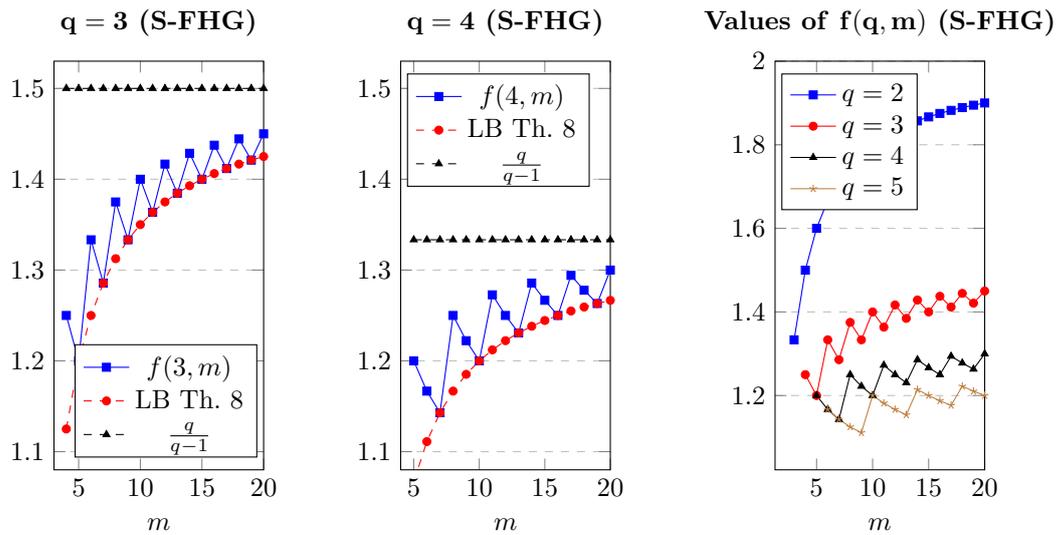
In fact, our proof does not only apply to S-FHGs, but to all symmetric  $\alpha$ -hedonic games. The more general result that we are able to show is that every  $q$ -size core stable outcome in an S- $\alpha$ HG is  $(m, f(q, m))$ -core stable with

$$f(q, m) = \max \left( 1, \left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{\alpha_m}{\alpha_q} + \frac{\mathbb{1}((m-1) \bmod (q-1)) \alpha_m}{\alpha((m-1) \bmod (q-1) + 1)} \right).$$

As we discuss in Section 3, this bound is equivalent to Equation (1) for S-FHGs. For S-ASHGs this implies a bound of  $f(q, m) = 1 + \left\lfloor \frac{m-2}{q-1} \right\rfloor$ .

■ **Table 1** For the given combinations of  $q$  and  $m$ , the table contains the value  $f(q, m)$  derived from Equation (1), such that a  $q$ -size core stable is  $(m, f(q, m))$ -stable in S-FHG.

$q \backslash m$	3	4	5	6	7	8	9	...	$\frac{q}{q-1}$
2	$\frac{4}{3}$	$\frac{6}{4}$	$\frac{8}{5}$	$\frac{10}{6}$	$\frac{12}{7}$	$\frac{14}{8}$	$\frac{16}{9}$		2
3	\	$\frac{5}{4}$	$\frac{6}{5}$	$\frac{8}{6}$	$\frac{9}{7}$	$\frac{11}{8}$	$\frac{12}{9}$	...	$\frac{3}{2}$
4	\	\	$\frac{6}{5}$	$\frac{7}{6}$	$\frac{8}{7}$	$\frac{10}{8}$	$\frac{11}{9}$		$\frac{4}{3}$



■ **Figure 2** Plotted values of  $f(q, m)$  for S-FHG such that every  $q$ -size core stable coalition structure is  $(m, f(q, m))$ -core stable. In the two leftmost figures, LB indicates the lower bound obtained in Theorem 8, while the black line indicates  $\frac{q}{q-1}$ , the limit of the upper bound.

Further, in Section 4 we derive lower bounds on these values as well, and show tightness for various combinations of  $(q, m)$ , and for various types of hedonic games. A summary of our results can be found in Table 3.

### 3 Main result

We begin with our main result, which quantifies the relationship between the two considered relaxed notions of core stability in symmetric  $\alpha$ -hedonic games. As the general proof for symmetric  $\alpha$ -hedonic games is quite notationally heavy, we defer the full proof to the supplementary material and only give the proof for symmetric fractional hedonic games here.

► **Theorem 2.** Any  $q$ -size core stable coalition structure  $\mathcal{C}$  in an  $S$ - $\alpha$ HG is  $(m, f(q, m))$ -core stable, for any integers  $m, q$  with  $m \geq q + 1$ , and  $f(q, m) =$

$$\max \left( 1, \left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{\alpha_m}{\alpha_q} + \frac{\mathbb{1}((m-1) \bmod (q-1)) \alpha_m}{\alpha((m-1) \bmod (q-1) + 1)} \right).$$

For fractional hedonic games, this reduces to:

► **Corollary 3.** For  $S$ -FHGs, every  $q$ -size core stable coalition structure  $\mathcal{C}$  is  $\left(m, 1 + \lfloor \frac{1}{q-1}(m-2) \rfloor\right)$ -core stable for any  $m \geq q + 1$ .

**Proof.** Consider a  $q$ -size core stable coalition structure  $\mathcal{C}$  and a coalition  $C$  of size  $m \geq q + 1$ . For a given coalition  $C' \subseteq C$  we let  $w(C') = \sum_{a_i \in C'} \left(u_i(C') - 2\frac{q}{q-1}u_i(\mathcal{C})\right)$  denote the *modified social welfare* of coalition  $C'$ . Let  $\mathcal{C}_{q-1}$  be the set of coalitions of size  $q - 1$  and consider the weighted hypergraph  $(C, \mathcal{C}_{q-1}, w)$ . Let  $M = \{C_1, \dots, C_{\lfloor \frac{m-1}{q-1} \rfloor}\}$  be any maximum weight, with regard to  $w$ , hypergraph matching, i.e., selection of non-overlapping sets from  $\mathcal{C}_{q-1}$ , of size  $\lfloor \frac{m-1}{q-1} \rfloor$  in this hypergraph. We note that a maximum weight hypergraph matching not of size  $\lfloor \frac{m-1}{q-1} \rfloor$  might have a larger weight, due to  $w$  being potentially negative. Let  $C_0$  be the set of unmatched agents by this hypergraph matching. The goal of our proof is now to show that there must be an unmatched agent who can only improve by a factor of at most  $1 + \lfloor \frac{1}{q-1}(m-2) \rfloor$ . The set  $C_0$  contains exactly  $(m - 1) \bmod (q - 1) + 1$  agents that are unmatched by  $M$ . Let  $a_0 \in C_0$ . For any  $i \in [\lfloor \frac{m-1}{q-1} \rfloor]$  we know that the coalition  $\{a_0\} \cup C_i$  is not  $q$ -size blocking. Thus, either one of the following two conditions has to hold:

- (i)  $\sum_{a_j \in C_i} u_0(a_j) \leq qu_0(\mathcal{C})$ ,
- (ii)  $\sum_{a_j \in C_i} u_0(a_j) > qu_0(\mathcal{C})$  and there is an  $a_\ell \in C_i$  with  $\sum_{a_j \in C_i \cup \{a_0\}} u_\ell(a_j) \leq qu_\ell(\mathcal{C})$

If we assume the latter scenario, we first notice that

$$\begin{aligned}
 \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} - 2\frac{q}{q-1}u_\ell(\mathcal{C}) &\leq \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} - 2 \sum_{a_j \in C_i \cup \{a_0\}} \frac{u_\ell(a_j)}{q-1} \\
 &= 2 \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} - 2 \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} - 2\frac{u_0(a_\ell)}{q-1} - \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} \\
 &= -2\frac{u_0(a_\ell)}{q-1} - \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} = 2 \sum_{a_j \in C_i \setminus \{a_\ell\}} \frac{u_0(a_j)}{q-1} - 2 \sum_{a_j \in C_i} \frac{u_0(a_j)}{q-1} - \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} \\
 &< 2 \sum_{a_j \in C_i \setminus \{a_\ell\}} \frac{u_0(a_j)}{q-1} - 2\frac{q}{q-1}u_0(\mathcal{C}) - \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} \\
 &= 2 \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(\frac{u_0(a_j)}{q-1}\right) - 2\frac{q}{q-1}u_0(\mathcal{C}) - \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(\frac{u_j(a_\ell)}{q-1}\right).
 \end{aligned}$$

For the first inequality, we used the second part of the assumption, while in the second-to-last line we used the first part of the assumption. Further, in the last equality, we used the symmetry of the utilities and the fact that  $u_\ell(a_\ell) = 0$ . Using this inequality, we can now obtain

$$\begin{aligned}
 &w((C_i \cup \{a_0\}) \setminus \{a_\ell\}) \\
 &= \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(u_j(C_i \cup \{a_0\} \setminus \{a_\ell\}) - 2\frac{q}{q-1}u_j(\mathcal{C})\right) \\
 &\quad + \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(\frac{u_0(a_j)}{q-1}\right) - 2\frac{q}{q-1}u_0(\mathcal{C}) \\
 &= \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(u_j(C_i) + \frac{u_j(a_0)}{q-1} - \frac{u_j(a_\ell)}{q-1} - 2\frac{q}{q-1}u_j(\mathcal{C})\right) \\
 &\quad + \sum_{a_j \in C_i \setminus \{a_\ell\}} \left(\frac{u_0(a_j)}{q-1}\right) - 2\frac{q}{q-1}u_0(\mathcal{C})
 \end{aligned}$$

$$\begin{aligned}
&= \sum_{a_j \in C_i \setminus \{a_\ell\}} \left( u_j(C_i) - 2 \frac{q}{q-1} u_j(\mathcal{C}) \right) \\
&\quad + 2 \sum_{a_j \in C_i \setminus \{a_\ell\}} \left( \frac{u_0(a_j)}{q-1} \right) - 2 \frac{q}{q-1} u_0(\mathcal{C}) - \sum_{a_j \in C_i \setminus \{a_\ell\}} \left( \frac{u_j(a_\ell)}{q-1} \right) \\
&> \sum_{a_j \in C_i \setminus \{a_\ell\}} \left( u_j(C_i) - 2 \frac{q}{q-1} u_j(\mathcal{C}) \right) + \sum_{a_j \in C_i} \frac{u_\ell(a_j)}{q-1} - 2 \frac{q}{q-1} u_\ell(\mathcal{C}) = w(C_i).
\end{aligned}$$

Hence, we get that  $w(C_i \cup \{a_0\} \setminus \{a_\ell\}) > w(C_i)$  and thus the hypergraph matching was not of maximum weight. Therefore,  $\sum_{a_j \in C_i} u_0(a_j) \leq q u_0(\mathcal{C})$  has to hold for every  $C_i$  and every  $a_0 \in C_0$ .

Next, if  $|C_0| = 1$  we know that  $(m-1)$  must be divisible by  $(q-1)$  and thus we can reformulate our bound as

$$1 + \frac{\lfloor \frac{1}{q-1}(m-2) \rfloor}{m} = 1 + \frac{\frac{1}{q-1}(m-q)}{m} = \frac{q(m-1)}{m(q-1)}.$$

Hence, by applying the previously calculated bound and by using the observation that  $|M| = \frac{m-1}{q-1}$  in the case, we obtain

$$\sum_{a_j \in \mathcal{C}} \frac{u_0(a_j)}{m} \leq \sum_{C_i \in M} \frac{q}{m} u_0(\mathcal{C}) = u_0(\mathcal{C}) \frac{q(m-1)}{m(q-1)},$$

which implies the result in case  $|C_0| = 1$ .

If  $|C_0| > 1$ , there has to be at least one agent  $a_0$  in  $C_0$  with  $\sum_{a_i \in C_0} u_0(a_i) \leq |C_0| u_0(\mathcal{C}) = ((m-1) \bmod (q-1) + 1) u_0(\mathcal{C})$ , since the set  $C_0$  of unmatched agents is non-blocking. Hence, we obtain that

$$\begin{aligned}
\sum_{a_j \in \mathcal{C}} \frac{u_0(a_j)}{m} &= \sum_{C_i \in M} \sum_{a_j \in C_i} \frac{u_0(a_j)}{m} + \sum_{a_i \in C_0} \frac{u_0(a_i)}{m} \\
&\leq \sum_{C_i \in M} \frac{q}{m} u_0(\mathcal{C}) + \frac{(m-1) \bmod (q-1) + 1}{m} u_0(\mathcal{C}) \\
&= \left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{q}{m} u_0(\mathcal{C}) + \frac{(m-1) \bmod (q-1) + 1}{m} u_0(\mathcal{C}) \\
&= \frac{u_0(\mathcal{C})}{m} \left( \left\lfloor \frac{m-1}{q-1} \right\rfloor q + (m-1) \bmod (q-1) + 1 \right) \\
&= \frac{u_0(\mathcal{C})}{m} \left( \frac{(m-1) - (m-1) \bmod (q-1)}{q-1} (q-1) \right. \\
&\quad \left. + \frac{(m-1) - (m-1) \bmod (q-1)}{q-1} + (m-1) \bmod (q-1) + 1 \right) \\
&= \frac{u_0(\mathcal{C})}{m} \left( m + \frac{(m-1) - (m-1) \bmod (q-1)}{q-1} \right) = \frac{u_0(\mathcal{C})}{m} \left( m + \left\lfloor \frac{m-2}{q-1} \right\rfloor \right).
\end{aligned}$$

The last step holds since  $(m-1) \bmod (q-1) > 0$ . Thus, every  $q$ -size core stable coalition structure  $\mathcal{C}$  is

$$\left( m, 1 + \frac{\lfloor \frac{1}{q-1}(m-2) \rfloor}{m} \right) \text{-core stable.}$$

Finally, to see that this is equivalent to the bound in Theorem 2 we see that if  $(m - 1) \bmod (q - 1) = 0$  it holds that

$$\left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{\alpha_m}{\alpha_q} + \frac{\mathbb{1}((m-1) \bmod (q-1)) \alpha_m}{\alpha((m-1) \bmod (q-1) + 1)} = \frac{m-1}{q-1} \frac{q}{m} = 1 + \frac{\left\lfloor \frac{1}{q-1}(m-2) \right\rfloor}{m}$$

and if  $(m - 1) \bmod (q - 1) \neq 0$

$$\begin{aligned} \left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{\alpha_m}{\alpha_q} + \frac{\mathbb{1}((m-1) \bmod (q-1)) \alpha_m}{\alpha((m-1) \bmod (q-1) + 1)} &= \left\lfloor \frac{m-1}{q-1} \right\rfloor \frac{q}{m} + \frac{(m-1) \bmod (q-1) + 1}{m} \\ &= 1 + \frac{\left\lfloor \frac{1}{q-1}(m-2) \right\rfloor}{m}. \end{aligned}$$

As a corollary we obtain an answer to the conjecture of [20], by confirming that every  $q$ -size core stable outcome is also  $\frac{q}{q-1}$ -improvement stable.

► **Corollary 4.** *For S-FHG, every  $q$ -size core stable coalition structure  $\mathcal{C}$  is  $\frac{q}{q-1}$ -improvement stable.*

**Proof.** This result follows from the observation that

$$1 + \frac{\left\lfloor \frac{1}{q-1}(m-2) \right\rfloor}{m} \leq 1 + \frac{\frac{1}{q-1}(m-2)}{m} \leq 1 + \frac{1}{q-1} = \frac{q}{q-1}$$

holds for all  $m$ . Thus, there is no coalition of size  $m > q$  in which every agent improves by a factor of more than  $\frac{q}{q-1}$ . ◀

If we restrict ourselves to the case of *simple fractional hedonic games (SS-FHG)*, i.e., S-FHG with binary utilities, we can show that this bound is not tight. This proof further provides a strengthening of Theorem 2 by [20].

► **Theorem 5.** *For every simple symmetric fractional hedonic game, any 3-size core stable coalition structure  $\mathcal{C}$  is*

$$\left( m, \frac{3}{2} \frac{(m-1)}{m} \right)\text{-core stable,}$$

for any integer  $m \geq 4$ .

**Proof.** We only prove the result for even values of  $m$ , as the result for odd  $m$  follows from the general result of Corollary 3. Consider a 3-size core stable coalition structure  $\mathcal{C}$  and a blocking coalition  $C$  of size  $m \geq 4$ . Further, we assume that every agent in  $C$  improves by more than a factor of  $k$ . Because all agents experience a strict improvement by forming  $C$ , each agent should be adjacent to at least one edge in the related graph  $G(C)$ . Since 3-size core stability implies 2-size core stability, there should be at least one agent  $a_i \in C$  for whom  $u_i(\mathcal{C}) \geq \frac{1}{2}$ . Since  $a_i$  improves by more than a factor of  $k$  it holds that

$$\sum_{a_\ell \in C \setminus \{a_i\}} u(i, \ell) > k \cdot m \cdot u_i(\mathcal{C}) \geq \frac{m}{2}, \quad (2)$$

which implies that  $a_i$  should be adjacent to at least  $\frac{m}{2} + 1$  edges in  $G(C)$ . Denote the set of agents that are connected to  $a_i$  through these edges by  $C' \subset C$ . For any triplet of agents  $\{a_i, a_j, a_k\}$  with  $\{a_j, a_k\} \subset C'$ , the 3-size core stability of  $\mathcal{C}$  implies that either  $u_i(\mathcal{C}) \geq \frac{2}{3}$ ,  $u_j(\mathcal{C}) \geq \frac{1}{2}$ , or  $u_k(\mathcal{C}) \geq \frac{1}{2}$  must hold. If  $u_i(\mathcal{C}) \geq \frac{2}{3}$  holds, then because  $a_i$  improves in  $C$  by a factor of more than  $k$  we get that

$$\frac{2}{3}k \leq k \cdot u_i(\mathcal{C}) < \frac{1}{m} \sum_{a_\ell \in C' \setminus \{a_i\}} u(i, \ell) \leq \frac{m-1}{m}. \quad (3)$$

Thus, we get that  $k < \frac{3}{2} \frac{m-1}{m}$  in this case. Alternatively, assume without loss of generality that  $u_j(\mathcal{C}) \geq \frac{1}{2}$ . Following the logic from Equation (2),  $a_j$  should be adjacent to at least  $\frac{m}{2} + 1$  edges in  $G(\mathcal{C})$ . This implies that there exists an agent  $a_\ell \in C'$  such that the agents  $\{a_i, a_j, a_\ell\}$  form a triangle, and hence at least one of these three agents should experience a utility of at least  $\frac{2}{3}$  in  $\mathcal{C}$ . Thus, Equation (3) holds for this agent and the result follows. ◀

Finally, as a second corollary of Theorem 2, we also obtain a bound for s-ASHGs.

► **Corollary 6.** *For S-ASHGs, every  $q$ -size core stable coalition structure  $\mathcal{C}$  is  $(m, 1 + \lfloor \frac{m-2}{q-1} \rfloor)$ -core stable for any  $m > q$ .*

## 4 Lower Bounds

Next, we focus on proving lower bounds for our setting. We first show a lower bound for the following subclass of S- $\alpha$ HGs, which includes S-FHGs, S-MFHGs, and S-ASHGs.

► **Definition 7.** *A function  $\alpha: [n] \rightarrow \mathbb{R}$  is hospitable if  $\frac{\alpha_q}{\alpha_{q-1}} \geq \frac{q-2}{q-1}$  for all integers  $q \geq 2$ . Accordingly, an S- $\alpha$ HG is hospitable if  $\alpha$  is hospitable.*

The intuition behind hospitable S- $\alpha$ HGs is that the utility of an agent in a coalition of size  $q-1$  will never decrease with more than a factor  $\frac{q-2}{q-1}$  when an additional agent is added to that coalition. This class includes ASHG, FHG, and MFHG. First, we can show that the bound derived in Theorem 2 is tight for hospitable S- $\alpha$ HGs when  $(m-1) \bmod (q-1) = 0$ .

► **Theorem 8.** *For any hospitable  $\alpha$ , there exists an instance of an S- $\alpha$ HG that contains a  $q$ -size core stable coalition structure  $\mathcal{C}$  which is not  $(m, \delta)$ -core stable for any  $\delta < \frac{\alpha_m(m-1)}{\alpha_q(q-1)}$ , with  $q, m \in \mathbb{N}$  and  $m \geq q+1$ .*

**Proof.** We construct an instance of an S- $\alpha$ HG that is  $q$ -size core stable, but which allows for a blocking coalition of size  $m \geq q+1$  in which all agents improve with at least a factor  $(\frac{\alpha_m(m-1)}{\alpha_q(q-1)})$ . Given a coalition structure  $\mathcal{C}$  in which  $u_i(\mathcal{C}) = 1$  for all agents  $i$ , and a blocking coalition  $C$  of size  $m$ , let  $u(i, j) = \frac{1}{\alpha_q(q-1)}$  for all agent pairs  $\{i, j\} \subset C$ . The resulting S- $\alpha$ HG is  $q$ -size core stable, since for any coalition  $C'$  of size at most  $q$  the utility of agent  $i$  in  $C'$  is at most

$\frac{(|C'|-1)\alpha_{|C'|}}{\alpha_q(q-1)} \leq \frac{\alpha_q(q-1)}{\alpha_q(q-1)} = 1$ , where the inequality is implied by recursively applying the definition of a hospitable S- $\alpha$ HG. As  $u_i(\mathcal{C}) = 1$  and since the utility of agent  $i$  in  $C$  is  $\frac{\alpha_m(m-1)}{\alpha_q(q-1)}$  for all agents  $i \in C$ , this implies that the coalition structure  $\mathcal{C}$  is not  $(m, \delta)$ -core stable for any  $\delta < \frac{\alpha_m(m-1)}{\alpha_q(q-1)}$ . ◀

Since if  $(m-1) \bmod (q-1) = 0$  it holds that  $\lfloor \frac{m-1}{q-1} \rfloor = \frac{m-1}{q-1}$  and  $\mathbf{1}((m-1) \bmod (q-1)) \alpha_m = 0$ , this implies that the bound obtained in Corollary 3 is tight for  $(m-1) \bmod (q-1) = 0$  and thus also for  $q = 2$ . Figure 2 illustrates the tightness of this lower bound for S-FHGs. Further, we show tightness of the bound in Theorem 2 for hospitable S- $\alpha$ HGs when  $q = 3$ .

► **Theorem 9.** *For any hospitable  $\alpha$ , there exists an instance of an S- $\alpha$ HG that contains a 3-size core stable coalition structure  $\mathcal{C}$  which is not  $(m, \delta)$ -core stable for any  $\delta < f(3, m)$ , with  $q, m \in \mathbb{N}$  and  $m \geq 4$ .*

## 41:10 Relaxed Core Stability for Hedonic Games with Size-Dependent Utilities

**Proof.** Note that when  $f(3, m) = 1$ , the result follows directly. Moreover, when  $m$  is odd, so  $(m-1) \bmod 2 = 0$ , the result follows from Theorem 8. When  $m$  is even and when  $f(3, m) > 1$ , we first observe that

$$f(3, m) = \left\lfloor \frac{m-1}{2} \right\rfloor \frac{\alpha_m}{\alpha_3} + \frac{\mathbb{1}((m-1) \bmod 2) \alpha_m}{\alpha((m-1) \bmod 2 + 1)} = \frac{m-2}{2} \frac{\alpha_m}{\alpha_3} + \frac{\alpha_m}{\alpha_2}.$$

Now we assume that we are given  $m$  agents  $c_1, \dots, c_m$  with  $u_i(\mathcal{C}) = 1$  for each  $c_i$ . We partition the agents into two sets  $C_1, C_2$  with  $C_1 = \{c_1, \dots, c_{\frac{m}{2}}\}$  and  $C_2 = \{c_{\frac{m}{2}+1}, \dots, c_m\}$ . If two agents  $c_i$  and  $c_j$  are in the same set, we define  $u(i, j) = \frac{1}{\alpha_3} - \frac{1}{\alpha_2}$ . Otherwise, we set  $u(i, j) = \frac{1}{\alpha_2}$ .

For any two agents  $\{c_i, c_j\}$  it thus holds that  $u_i(\{c_i, c_j\}) \leq \alpha_2 \max(\frac{1}{\alpha_3} - \frac{1}{\alpha_2}, \frac{1}{\alpha_2}) = \max(\frac{\alpha_2}{\alpha_3} - 1, 1) \leq 1$  and, therefore, these two agents do not form a blocking coalition. Further, for any three agents  $\{c_i, c_j, c_k\}$  we have two cases: (i) either all three agents come from the same set, then we get that  $u_i(\{c_i, c_j, c_k\}) = \frac{2\alpha_3}{\alpha_3} - \frac{2\alpha_3}{\alpha_2} = 2 - \frac{2\alpha_3}{\alpha_2} \leq 2 - \frac{\alpha_3}{\alpha_3} = 1$ ; (ii) one agent (without loss of generality  $c_k$ ) has to be from a different partition than the other two; then it holds that  $u_i(\{c_i, c_j, c_k\}) = \frac{\alpha_3}{\alpha_3} - \frac{\alpha_3}{\alpha_2} + \frac{\alpha_3}{\alpha_2} = 1$ . Hence, this coalition is 3-stable.

Finally, we get that

$$u_i(\{c_1, \dots, c_m\}) = \alpha_m \left( \frac{m-2}{2} \left( \frac{1}{\alpha_3} - \frac{1}{\alpha_2} \right) + \frac{m}{2} \frac{1}{\alpha_2} \right) = \frac{m-2}{2} \frac{\alpha_m}{\alpha_3} + \frac{\alpha_m}{\alpha_2} = f(3, m). \blacktriangleleft$$

Lastly, we provide additional tightness results of the bound in Theorem 2 for S-FHG and S-ASHG. We defer the proofs of Theorems 11 and 12 to the supplementary material.

► **Theorem 10.** *There exists a  $q$ -size core stable coalition structure  $\mathcal{C}$  in an S-FHG which is not  $(q+1, \delta)$ -core stable for any  $\delta < \frac{q+2}{q+1}$ .*

**Proof.** Assume we are given  $q+1$  agents  $a_1, \dots, a_{q+1}$  with  $u_i(\mathcal{C}) = 1$ . Let the edge weights be such that the edges with weight two form a cycle and all other edges have weight one, i.e., let  $u(i, j) = 2$  for all  $(i, j) \in \mathcal{C}$  for which  $j = i+1$ , let  $u(n, 1) = 2$ , and let  $u(k, l) = 1$  for all other edges. Note that in each subset  $C \subset \mathcal{C}$  with  $|C| < q+1$  there is at least one agent who is adjacent to at most one edge of weight two to the other agents in  $C$ , because the edges with weight two form a cycle over all  $q+1$  agents. Hence, for each subset  $C \subset \mathcal{C}$  with  $|C| < q+1$  there is at least one agent with a utility of at most  $\frac{2+|C|-2}{|C|} = 1$ , which implies that  $\mathcal{C}$  is  $q$ -size core stable. Furthermore, the coalition of all  $q+1$  agents offers a utility of  $\frac{2 \cdot 2 + q - 2}{q+1} = \frac{q+2}{q+1}$ . As a result,  $\mathcal{C}$  is not  $(q+1, \frac{q+2}{q+1} - \varepsilon)$ -core stable for any  $\varepsilon > 0$ . ◀

► **Theorem 11.** *There exists a  $4$ -size core stable coalition structure  $\mathcal{C}$  in an S-FHG which is not  $(m, \delta)$ -core stable, for any  $\delta < 1 + \frac{\lfloor \frac{1}{3}(m-2) \rfloor}{m}$ , and for any integer  $m \geq 5$ .*

► **Theorem 12.** *There exists a  $4$ -size core stable coalition structure  $\mathcal{C}$  in an S-ASHG which is not  $(m, \delta)$ -core stable, for any  $\delta < 1 + \lfloor \frac{m-2}{3} \rfloor$ , and for any integer  $m \geq 5$ .*

While we were not able to show the tightness of Theorem 2 for other values of  $\alpha$ ,  $q$ , and  $m$  than the ones described in this section, we found some examples to show the tightness of the result for additional values of  $(q, m)$  that are not covered by Theorems 8-12 through the use of the integer linear programming approach described in the supplementary material, as shown in Table 2. A summary of our results can be found in Table 3.

■ **Table 2** Additional values of  $(q, m)$  for which we found instances proving the tightness of the bound in Theorem 2 by using the integer linear programming approach described in the supplementary material.

<b>S-FHG</b>	$(5, m \leq 8), (6, m \leq 10), (7, m \leq 10),$ $(8, m \leq 11)$
<b>S-ASHG</b>	$(5, m \leq 8), (6, m \leq 10), (7, m \leq 12),$ $(8, m \leq 13), (9, m \leq 13), (10, m \leq 13),$ $(11, m \leq 13), (12, 13)$

■ **Table 3** Summary and tightness results for the values of  $f(q, m)$  such that every  $q$ -size core stable coalition structure is  $(q, f(q, m))$ -stable, with  $m \geq q + 1$ , for various types of hedonic games. A ✓ indicates that we were able to show tightness of  $\mathbf{f}(\mathbf{q}, \mathbf{m})$  for this type of hedonic game, while a ✗ indicates that this tightness is still open.

Hedonic Game	$\mathbf{f}(\mathbf{q}, \mathbf{m})$	Tightness proof for...			
		$(q - 1) \mid (m - 1)$	$q = 3$	$q = 4$	Other values of $(q, m)$
S- $\alpha$ HG	see Theorem 2	✗	✗	✗	$\emptyset$
Hospitable S- $\alpha$ HG	see Theorem 2	✓	✓	✗	$\emptyset$
S-FHG	$1 + \frac{1}{m} \lfloor \frac{m-2}{q-1} \rfloor$	✓	✓	✓	$(q, q + 1)$ & Table 2
S-MFHG	1	✓	✓	✓	all combinations
S-ASHG	$1 + \lfloor \frac{m-2}{q-1} \rfloor$	✓	✓	✓	Table 2

## 5 Efficiency

In this section, we study the price of anarchy for core-relaxations of symmetric  $\alpha$ -hedonic games. Using the same notation as Fanelli et al. [20], we denote the *social welfare* of a coalition structure  $\mathcal{C}$  by  $SW(\mathcal{C}) = \sum_{i \in A} u_i(\mathcal{C})$ , which is simply the sum of the agents' utilities. Moreover, let  $\mathcal{G} = (\alpha, A, u)$  represent an instance of an S- $\alpha$ HG. Given an S- $\alpha$ HG  $\mathcal{G}$ , let  $q$ -SIZE CORE( $\mathcal{G}$ ) be the set of  $q$ -size core stable coalition structures, and let  $k$ -IMPR CORE( $\mathcal{G}$ ) be the set of  $k$ -improvement core stable coalition structures. We define the  *$q$ -size core price of anarchy* of an S- $\alpha$ HG  $\mathcal{G}$  as the ratio between the social welfare of the coalition structure  $\mathcal{C}^*(\mathcal{G})$  that maximizes social welfare, and that of the  $q$ -size core stable coalition structure with the worst social welfare, i.e.,  $q$ -SIZE CPOA( $\mathcal{G}$ ) =  $\max_{\mathcal{C} \in q\text{-SIZE CORE}(\mathcal{G})} \frac{SW(\mathcal{C}^*(\mathcal{G}))}{SW(\mathcal{C})}$ . Similarly, we define the  *$k$ -improvement core price of anarchy* as  $k$ -IMPR CPOA( $\mathcal{G}$ ) =  $\max_{\mathcal{C} \in k\text{-IMPR CORE}(\mathcal{G})} \frac{SW(\mathcal{C}^*(\mathcal{G}))}{SW(\mathcal{C})}$ .

Using Corollary 3, we can extend the results by Fanelli et al. [20] about the  $q$ -size core price of anarchy for S-FHGs.

► **Corollary 13.** *For any S-FHG  $\mathcal{G}$ , it holds that  $q$ -SIZE CPOA( $\mathcal{G}$ )  $\leq \frac{2q}{q-1}$ , for any integer  $q \geq 2$ , and this bound is tight.*

**Proof.** By Corollary 4, we know that the social welfare of the worst  $q$ -size core stable coalition structure is at least the social welfare of the worst  $\frac{q}{q-1}$ -improvement core stable coalition structure. Moreover, by Theorem 8 by [20], we know that the  $k$ -improvement CPOA of an S-FHG is upper bounded by  $2k$ , for any  $k \geq 1$ . The tightness of the bound follows from Theorem 9 by [20]. ◀

Next, we show upper bounds on the  $q$ -SIZE CPOA( $\mathcal{G}$ ) and the  $k$ -IMPR CPOA( $\mathcal{G}$ ) for the following subclass of S- $\alpha$ HGs.

► **Definition 14.** A function  $\alpha: [n] \rightarrow \mathbb{R}$  is decreasing if  $\alpha_q \geq \alpha_{q+1}$  for all integers  $q \geq 1$ . Accordingly, an S- $\alpha$ HG is decreasing if  $\alpha$  is decreasing.

The class of decreasing S- $\alpha$ HGs is distinct from the class of hospitable S- $\alpha$ HGs defined in Definition 7, but it also contains S-FHGs, S-ASHGs, and S-MFHGs. When restricting our focus to the subclass of decreasing S- $\alpha$ HGs, we can generalize Theorem 8 by [20] to obtain the same upper bound on the  $k$ -improvement core price of anarchy.

► **Theorem 15.** For any decreasing S- $\alpha$ HG  $\mathcal{G}$  and for every  $k \geq 1$ ,  $k$ -IMPR  $\text{CPOA}(\mathcal{G}) \leq 2k$ .

**Proof.** The proof is identical to the proof of Theorem 8 by [20]. Using their notation, with the adapted definition that  $\mu_i^>(C) = \alpha(|C|) \cdot \delta_C^>(i)$ , the only required alteration to their proof is that equation (2) in their proof should be replaced by:

$$\mu_{i_t}(C_t^*) = \alpha(|C_t^*|) \cdot \delta_{C_t^*}^>(i_t) \geq \alpha(|C^*|) \cdot \delta_{C^*}^>(i_t) = \mu_{i_t}^>(C^*),$$

which holds, by definition, because we are only considering decreasing S- $\alpha$ HGs. ◀

Lastly, we can use a similar reasoning as in the proof of Corollary 13 and use the bounds from Theorems 3 and 15 to obtain a general upper bound on the  $q$ -size core price of anarchy for decreasing S- $\alpha$ HGs.

► **Theorem 16.** For any decreasing S- $\alpha$ HG  $\mathcal{G}$ ,  $q$ -SIZE  $\text{CPOA}(\mathcal{G}) \leq 2 \cdot \max_{m \geq q+1} f(m, q)$ .

Note that this result implies a core price of anarchy of 2 for S-MFHGs, where the core price of anarchy of an S- $\alpha$ HG  $\mathcal{G}$  is simply defined as  $\max_q q$ -SIZE  $\text{CPOA}(\mathcal{G})$ . As such, we answered an open question by [26], who found a lower bound on the core price of stability of 2 and an upper bound for the core price of anarchy of 4 in S-MFHGs.

► **Corollary 17.** For any S-MFHG, the core price of anarchy is upper bounded by 2.

## 6 Conclusion and Outlook

In our paper, we studied hedonic games and the relationship between different relaxed notions of core stability. Most importantly, for a large class of hedonic games, we obtained a general upper bound  $f(q, m)$  such that every  $q$ -size core stable outcome is  $(m, f(q, m))$ -core stable. That is, a coalition of size  $m$  can deviate at most by a factor of  $f(q, m)$ . Our bound also allows us to answer a conjecture by [20] that every  $q$ -size core stable outcome in symmetric fractional hedonic games is  $\frac{q}{q-1}$ -improvement core stable. Finally, we also obtain some lower bounds. However, even for fractional and additively separable hedonic games, our bounds are not tight yet. For both, we were only able to show the tightness up to  $q = 4$ . The smallest case which is unknown (both for fractional and additively separable hedonic games) is the tightness for  $q = 5$  and  $m = 10$ , see Table 2 and Theorem 8. For both kinds of hedonic games, our integer linear programming approach was not able to construct a counterexample, nor show that no counterexample exists. Thus, improving our lower bounds seems like a challenging and interesting venue for future work. Further, it would be interesting to see if the generalization of  $\alpha$ -hedonic games, could see application in other areas of hedonic games as well. For instance, it might be interesting to classify for which  $\alpha$ -hedonic games certain dynamics converge (see for instance Boehmer et al. [7].)

## References

- 1 H. Aziz and R. Savani. Hedonic games. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 15. Cambridge University Press, 2016.
- 2 Haris Aziz, Florian Brandl, Felix Brandt, Paul Harrenstein, Martin Olsen, and Dominik Peters. Fractional hedonic games. *ACM Transactions on Economics and Computation (TEAC)*, 7(2):1–29, 2019.
- 3 Haris Aziz, Felix Brandt, and Hans Georg Seedig. Computing desirable partitions in additively separable hedonic games. *Artificial Intelligence*, 195:316–334, 2013.
- 4 Haris Aziz, Serge Gaspers, Joachim Gudmundsson, Julián Mestre, and Hanjo Taubig. Welfare maximization in fractional hedonic games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 461–467, 2015.
- 5 Vittorio Bilò, Angelo Fanelli, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli. Nash stable outcomes in fractional hedonic games: Existence, efficiency and computation. *Journal of Artificial Intelligence Research*, 62:315–371, 2018.
- 6 Vittorio Bilò, Gianpiero Monaco, and Luca Moscardelli. Hedonic games with fixed-size coalitions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9287–9295, 2022.
- 7 Niclas Boehmer, Martin Bullinger, and Anna Maria Kerkmann. Causes of stability in dynamic coalition formation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. Forthcoming.
- 8 Niclas Boehmer and Edith Elkind. Individual-based stability in hedonic diversity games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34(2), pages 1822–1829, 2020.
- 9 A. Bogomolnaia and M. O. Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002.
- 10 Felix Brandt and Martin Bullinger. Finding and recognizing popular coalition structures. *Journal of Artificial Intelligence Research*, 74:569–626, 2022.
- 11 Felix Brandt, Martin Bullinger, and Leo Tappe. Single-agent dynamics in additively separable hedonic games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(5), pages 4867–4874, June 2022. doi:10.1609/aaai.v36i5.20415.
- 12 Felix Brandt, Martin Bullinger, and Anaëlle Wilczynski. Reaching individually stable coalition structures in hedonic games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35(6), pages 5211–5218, 2021.
- 13 Robert Brederick, Edith Elkind, and Ayumi Igarashi. Hedonic diversity games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 565–573, 2019.
- 14 Martin Bullinger and Stefan Kober. Loyalty in cardinal hedonic games. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 66–72, 2021.
- 15 Raffaello Carosi, Gianpiero Monaco, and Luca Moscardelli. Local core stability in simple symmetric fractional hedonic games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 574–582, 2019.
- 16 Jiehua Chen, Gergely Csáj, Sanjukta Roy, and Sofia Simola. Cores in friend-oriented hedonic games: Verification is surprisingly harder than searching. In *Proceedings of the 22nd International Conference on Autonomous Agents and MultiAgent Systems*, 2023. Forthcoming.
- 17 Jiehua Chen and Sanjukta Roy. Multi-dimensional stable roommates in 2-dimensional euclidean space. In *30th Annual European Symposium on Algorithms, ESA 2022*, 2022.
- 18 Agnes Cseh, Tamás Fleiner, and Petra Harján. Pareto optimal coalitions of fixed size. *Journal of Mechanism and Institution Design*, 4:1, 2019.
- 19 J. H. Drèze and J. Greenberg. Hedonic coalitions: Optimality and stability. *Econometrica*, 48(4):987–1003, 1980.

- 20 Angelo Fanelli, Gianpiero Monaco, and Luca Moscardelli. Relaxed core stability in fractional hedonic games. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 182–188, 2021.
- 21 Robert Galian, Thekla Hamm, Dusan Knop, Simon Schierreich, and Ondrej Suchý. Hedonic diversity games: A complexity picture with more than two colors. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 5034–5042, 2022.
- 22 Tesshu Hanaka and Michael Lampis. Hedonic games and treewidth revisited. In *30th Annual European Symposium on Algorithms (ESA 2022)*, 2022.
- 23 Anna Maria Kerkmann, Nhan-Tam Nguyen, Anja Rey, Lisa Rey, Jörg Rothe, Lena Schend, and Alessandra Wiechers. Altruistic hedonic games. *Journal of Artificial Intelligence Research*, 75:129–169, 2022.
- 24 Lily Li, Evi Micha, Aleksandar Nikolov, and Nisarg Shah. Partitioning friends fairly. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. Forthcoming.
- 25 Michael McKay and David Manlove. The three-dimensional stable roommates problem with additively separable preferences. In *Algorithmic Game Theory: 14th International Symposium, SAGT 2021, Proceedings*, pages 266–280. Springer, 2021.
- 26 Gianpiero Monaco, Luca Moscardelli, and Yllka Velaj. Stable outcomes in modified fractional hedonic games. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–29, 2020.
- 27 Martin Olsen. On defining and computing communities. In *Proceedings of the Eighteenth Computing: The Australasian Theory Symposium-Volume 128*, pages 97–102, 2012.

# Recontamination Helps a Lot to Hunt a Rabbit

Thomas Dissaux ✉

Université Côte d’Azur, Inria, CNRS, I3S, France

Foivos Fioravantes ✉ 🏠 

Université Côte d’Azur, Inria, CNRS, I3S, France

Department of Theoretical Computer Science, FIT, Czech Technical University in Prague, Czech Republic

Harmender Gahlawat ✉ 🏠 

Ben-Gurion University of the Negev, Beersheba, Israel

Nicolas Nisse ✉ 🏠

Université Côte d’Azur, Inria, CNRS, I3S, France

---

## Abstract

The HUNTERS AND RABBIT game is played on a graph  $G$  where the Hunter player shoots at  $k$  vertices in every round while the Rabbit player occupies an unknown vertex and, if it is not shot, must move to a neighbouring vertex after each round. The Rabbit player wins if it can ensure that its position is never shot. The Hunter player wins otherwise. The hunter number  $h(G)$  of a graph  $G$  is the minimum integer  $k$  such that the Hunter player has a winning strategy (i.e., allowing him to win whatever be the strategy of the Rabbit player). This game has been studied in several graph classes, in particular in bipartite graphs (grids, trees, hypercubes...), but the computational complexity of computing  $h(G)$  remains open in general graphs and even in more restricted graph classes such as trees. To progress further in this study, we propose a notion of monotonicity (a well-studied and useful property in classical pursuit-evasion games such as Graph Searching games) for the HUNTERS AND RABBIT game imposing that, roughly, a vertex that has already been shot “must not host the rabbit anymore”. This allows us to obtain new results in various graph classes.

More precisely, let the monotone hunter number  $mh(G)$  of a graph  $G$  be the minimum integer  $k$  such that the Hunter player has a monotone winning strategy. We show that  $pw(G) \leq mh(G) \leq pw(G) + 1$  for any graph  $G$  with pathwidth  $pw(G)$ , which implies that computing  $mh(G)$ , or even approximating  $mh(G)$  up to an additive constant, is NP-hard. Then, we show that  $mh(G)$  can be computed in polynomial time in split graphs, interval graphs, cographs and trees. These results go through structural characterisations which allow us to relate the monotone hunter number with the pathwidth in some of these graph classes. In all cases, this allows us to specify the hunter number or to show that there may be an arbitrary gap between  $h$  and  $mh$ , i.e., that monotonicity does not help. In particular, we show that, for every  $k \geq 3$ , there exists a tree  $T$  with  $h(T) = 2$  and  $mh(T) = k$ . We conclude by proving that computing  $h$  (resp.,  $mh$ ) is FPT parameterised by the minimum size of a vertex cover.

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

**Keywords and phrases** Hunter and Rabbit, Monotonicity, Graph Searching

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.42

**Related Version** *Full Version*: <https://hal.science/hal-03995642v3> [12]

**Funding** This research was partially funded by the project UCA JEDI (ANR-15-IDEX-01) and EUR DS4H Investments in the Future (ANR-17-EURE-004) and the ANR Digraphs and the IFCAM project Applications of graph homomorphisms (MA/IFCAM/18/39) and ERC grant titled PARAPATH and the CTU Global postdoc fellowship program.



© Thomas Dissaux, Foivos Fioravantes, Harmender Gahlawat, and Nicolas Nisse; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 42; pp. 42:1–42:14

Leibniz International Proceedings in Informatics



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

## 1 Introduction

The HUNTERS AND RABBIT game is played on a graph  $G$  and with a positive integer  $k$  (the number of hunters), where the Hunter player shoots at  $k$  vertices in every round while the Rabbit player occupies an unknown vertex and, if it is not shot, must move to a neighbouring vertex after each round. The Rabbit player wins if he can ensure that its position is never shot. The Hunter player wins otherwise. The HUNTERS AND RABBIT game was first introduced in [7], in the case  $k = 1$ , where it was shown that the Hunter player wins in a tree  $T$  if and only if  $T$  does not contain as subgraph any tree obtained from a star with 3 leaves by subdividing each edges twice. This result was also observed in [19], where the authors also consider the minimum number of rounds needed for the Hunter player to win. The version where  $k > 1$  was first considered in [1]. Observe that, if  $k = |V(G)| - 1$ , the Hunter player can win in any graph  $G$  (in two rounds) by shooting twice a subset of  $k$  vertices of  $G$ . Hence, let the *hunter number* of  $G$ , denoted by  $h(G)$ , be the minimum integer  $k$  such that  $k$  hunters can win in  $G$  whatever be the rabbit strategy.

In [1], it is shown that the hunter number is closed under taking subgraphs. Moreover, this result trivially extends to the case when the starting positions of the rabbit are restricted. The exact value of  $h(G)$  has been determined for several specific families of graphs  $G$ . For any  $n \geq 2$ ,  $h(P_n) = 1$  where  $P_n$  is the path with  $n$  vertices [7] (because the rabbit is forced to move at every round,  $h(P_1) = 0$ ). For any  $n \geq 3$ ,  $h(C_n) = 2$  and  $h(K_n) = n - 1$ , where  $C_n$  and  $K_n$  are the cycle and complete graph on  $n$  vertices respectively [1]. Moreover,  $h(G_{n \times m}) = \lfloor \frac{\min\{n,m\}}{2} \rfloor + 1$  [1] and  $h(Q^n) = 1 + \sum_{i=0}^{n-2} \binom{i}{\lfloor i/2 \rfloor}$  [5], where  $G_{n \times m}$  is the  $n \times m$  grid and  $Q^n$  is the hypercube with dimension  $n$ . The case of bipartite graphs has been particularly studied because it was proved in [1] that we may constrain the rabbit to start in a fixed part of the bipartition, without decreasing the hunter number. By taking advantage of the bipartiteness of trees, it was proved that, for any tree  $T$ ,  $h(T) \leq \lceil \frac{1}{2} \log_2(|V(T)|) \rceil$  [17]. Surprisingly, the computational complexity of the problem that takes a graph  $G$  and an integer  $k$  as inputs and aims at deciding whether  $h(G) \leq k$  is still open, even if  $G$  is restricted to be a tree.

In this paper, we progress further in this research direction by exhibiting new classes of graphs  $G$  where  $h(G)$  can be determined in polynomial time.

**Graph Searching games.** The HUNTERS AND RABBIT game takes place in the larger class of Graph Searching games initially introduced in [6, 26]. In these pursuit-evasion games, one player plays with a team of searchers (or hunters, etc.) that must track a fugitive (or robber, rabbit, etc.) moving in a graph. Multiple games fall under this framework, each one specifying its own rules on, for example, the available moves of the searchers, the speed of the fugitive, whether the fugitive is visible or not, etc. Several variations of Graph Searching games have been studied in the literature due to their numerous applications in artificial intelligence [21], robot motion planning [9], constraint satisfaction problems and database theory [16], and distributed computing [25]. The study of such games also leads to significant implications in graph theory and algorithms. Indeed, many variants of these games provide algorithmic interpretations of width measures of graphs like treewidth [27], pathwidth [26], tree-depth [15], hypertree-width [2], cycle-rank [15], and directed tree-width [22]. Central to the connection between Graph Searching games and such structural parameters is the notion of *monotonicity* [3, 27, 24, 20]. In short, a searchers' strategy is *monotone* if it ensures that the fugitive can never "recontaminate" a vertex, i.e., it can never access a vertex that has already been "visited" by a searcher. The main question is then, given a game,

whether “recontamination does not help in this game” [23], i.e., whether there always exists, in this game, an optimal (in terms of number of searchers) monotone winning strategy for the searchers. In particular, the monotonicity played a central role in the proof that the minimum number of searchers to capture an invisible (resp., visible) fugitive in the node-searching game played in a graph  $G$  equals its pathwidth plus one [3] (resp., treewidth plus one [27]).

Unsurprisingly, the HUNTERS AND RABBIT game also has a close relationship with the pathwidth of graphs. Precisely, the hunter number of any graph is at most its pathwidth plus one [1]. In this paper, we investigate further this relationship and, for this purpose, we define, and study, a notion of monotonicity adapted to the HUNTERS AND RABBIT game.

**Our contribution.** In Section 2, we give the main notations used throughout this paper. In Section 3, we introduce the notion of monotonicity for the HUNTERS AND RABBIT game which is not straightforward; let  $mh(G)$  denote the *monotone hunter number* of  $G$ . Then, we prove that  $mh(G) \in \{pw(G), pw(G) + 1\}$  in any graph  $G$ . Along with implying that it is NP-complete to compute  $mh(G)$  for a graph  $G$ , this result also implies that it is NP-hard to approximate  $mh(G)$  up to an additive error of  $|V(G)|^\varepsilon$ , for  $0 < \varepsilon < 1$ . On the positive side, in Section 4, we give polynomial time algorithms to determine  $h(G)$  and/or  $mh(G)$  in cographs, split and interval graphs. In Section 5, we adapt the Parsons’ Lemma [26] to the case of the monotone HUNTERS AND RABBIT game which leads to a polynomial time algorithm that computes  $mh(T)$  for any tree  $T$ . Then, we investigate the monotonicity property in the case of the “bipartite” variant of the HUNTERS AND RABBIT game (see [1, 17]) in trees. In particular, we show that, for any  $k \in \mathbb{N}$ , there exist trees  $T$  such that  $h(T) = 2$  and  $mh(T) \geq k$ . That is, “recontamination helps a lot” in the HUNTERS AND RABBIT game. Finally, in Section 6, we show as a general positive result that the problem of deciding if  $h(G) \leq k$  or  $mh(G) \leq k$ , for some given integer  $k$ , is in FPT when parameterized by the vertex cover number of  $G$ . This is done through kernelization. We close our study by providing directions for further research in Section 7.

## 2 Preliminaries

Unless mentioned otherwise, in this paper we will always deal with graphs  $G = (V, E)$  that are non empty, finite, undirected, connected and simple. For any two adjacent vertices  $x, y \in V$ , let  $xy \in E$  denote the edge between  $x$  and  $y$ . Given a set  $S \subseteq V$ , let  $G[S]$  denote the subgraph of  $G$  induced by (the vertices in)  $S$  and let  $G \setminus S$  denote the subgraph  $G[V \setminus S]$ . For any  $v \in V$  and  $X \subseteq V$ , let  $N_X(v) = \{u \in X \mid uv \in E\}$  be the *open neighbourhood* of  $v$  in  $X$  and let the *closed neighbourhood* of  $v$  in  $X$  be  $N_X[v] = (N_X(v) \cup \{v\}) \cap X$ . If  $X = V$ , we simply write  $N(v)$  and  $N[v]$  respectively. For any  $S \subseteq V$ , let  $N(S) = \bigcup_{v \in S} N(v) \setminus S$  and  $N[S] = N(S) \cup S$ . The degree  $d(v) = |N(v)|$  is the number of neighbours of  $v$  and let  $\delta(G) = \min_{v \in V} d(v)$ . An *independent set* of a graph  $G = (V, E)$  is a subset  $I$  of  $V$  such that, for every  $u, v \in I$ ,  $uv \notin E$ . A graph is *bipartite* if its vertex-set can be partitioned into two independent sets.

**Hunters and Rabbit game.** The HUNTERS AND RABBIT game is played between two players, Hunter and Rabbit, on a graph. Let  $k \in \mathbb{N}^*$ . The Hunter player controls  $k$  hunters and the Rabbit player controls a single rabbit. First, the Rabbit player places the rabbit at a vertex  $r_0 \in V$ . The rabbit is *invisible*, i.e., the position of the rabbit is not known to the hunters. Then, the game proceeds in *rounds*. In each round  $i \geq 1$ , first, the Hunter player selects a non empty subset  $S_i \subseteq V$  of at most  $k$  vertices of  $G$  (we say that the vertices in  $S_i$

are *shot* at round  $i$ ). If the current position  $r_{i-1}$  of the rabbit is shot, i.e., if  $r_{i-1} \in S_i$  (we say that the rabbit is shot), then the Hunter player wins, and the game stops. Otherwise, the rabbit must move from its current position  $r_{i-1}$  to a vertex  $r_i \in N(r_{i-1})$ , and the next round starts. The Rabbit player wins if the rabbit avoids being shot forever.

A *hunter strategy* in  $G = (V, E)$  is a finite sequence  $\mathcal{S} = (S_1, \dots, S_\ell)$  of non empty subsets of vertices of  $G$ . Let  $h(\mathcal{S}) := \max_{1 \leq i \leq \ell} |S_i|$  and let us say that  $\mathcal{S}$  *uses*  $h(\mathcal{S})$  hunters. A *rabbit trajectory in  $G$  starting from  $W \subseteq V$*  ( $W$  will always be assumed non empty) is any walk  $(r_0, \dots, r_\ell)$  such that  $r_0 \in W$  and  $r_i \in N(r_{i-1})$  for every  $1 \leq i \leq \ell$ . A hunter strategy is *winning with respect to  $W$*  if, for every rabbit trajectory  $(r_0, \dots, r_\ell)$  starting from  $W$ , there exists  $0 \leq j < \ell$  such that  $r_j \in S_{j+1}$ , i.e, the rabbit is eventually shot whatever be its trajectory starting from  $W$ . Given a hunter strategy  $\mathcal{S} = (S_1, \dots, S_\ell)$ , a rabbit trajectory  $(r_0, \dots, r_\ell)$  starting from  $W$  is *winning against  $\mathcal{S}$*  if  $r_i \notin S_{i+1}$  for every  $0 \leq i < \ell$ . A *winning hunter strategy* is any winning hunter strategy with respect to  $V$  and a *rabbit trajectory* is any rabbit trajectory starting from  $V$ .

The *hunter number of  $G = (V, E)$  with respect to  $W \subseteq V$* , denoted by  $h_W(G)$ , is the minimum integer  $k$  such that there exists a winning hunter strategy with respect to  $W$  and using  $k$  hunters. Let  $h(G) = h_V(G)$  be the *hunter number of  $G$* . The Rabbit player has a *strategy  $\mathcal{R}$  starting from  $W \subseteq V$  against  $k \geq 1$  hunters* if, for every hunter strategy  $\mathcal{S}$  using  $k$  hunters, there exists a rabbit trajectory  $\mathcal{R}(\mathcal{S})$  that is winning against  $\mathcal{S}$ . If such a strategy  $\mathcal{R}$  exists, then  $h_W(G) > k$ .

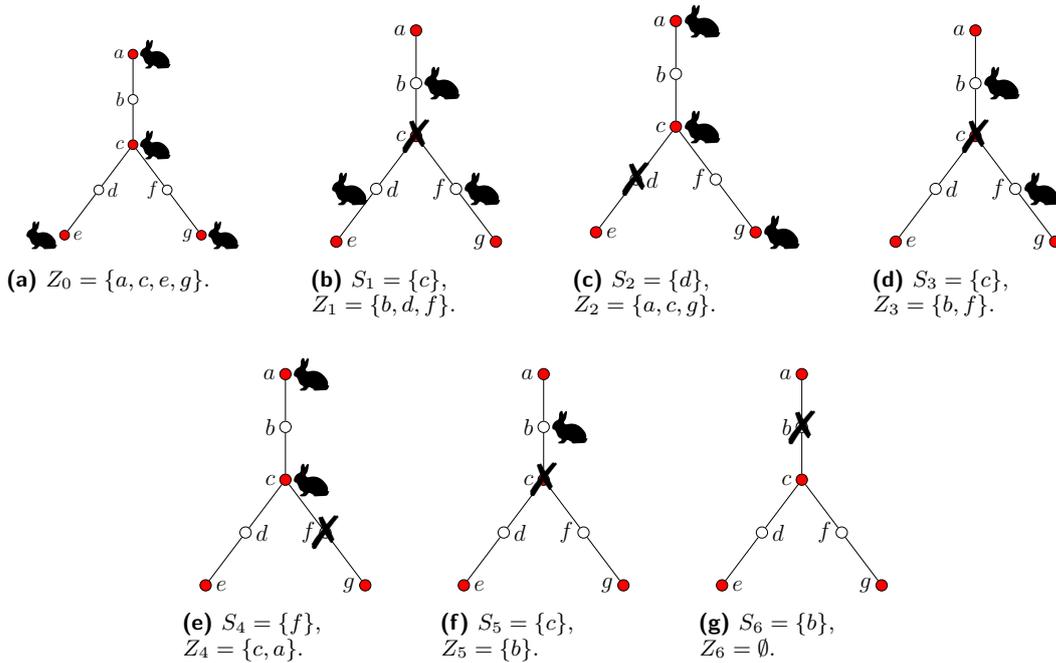
For any hunter strategy  $\mathcal{S} = (S_1, \dots, S_\ell)$ , it will be convenient to identify the potential positions of a rabbit (starting in  $W \subseteq V$ ) after each round. Precisely, let  $\mathcal{Z}^W(\mathcal{S}) = (Z_0^W(\mathcal{S}), \dots, Z_\ell^W(\mathcal{S}))$  be defined as follows. Let  $Z_0^W(\mathcal{S}) = W$  and, for every  $0 < i \leq \ell$ , let  $Z_i^W(\mathcal{S})$  be the set of vertices  $v$  such that there exists a rabbit trajectory  $(r_0, r_1, \dots, r_i = v)$  such that  $r_0 \in W$  and, for every  $0 \leq j < i$ ,  $r_j \notin S_{j+1}$ . Formally, for any  $1 \leq i \leq \ell$ , let  $Z_i^W(\mathcal{S}) = \{x \in V(G) \mid \exists y \in (Z_{i-1}^W(\mathcal{S}) \setminus S_i) \wedge (xy \in E(G))\}$ . Intuitively,  $Z_i^W(\mathcal{S})$  is the set of vertices that the rabbit (starting from some vertex in  $W$ ) can have reached at the end of the  $i^{\text{th}}$  round without having been shot. We will refer to the vertices in  $Z_i^W(\mathcal{S})$  as the *contaminated* vertices after round  $i$ . Note that, if  $\mathcal{S}$  is winning, then  $Z_\ell^W(\mathcal{S}) = \emptyset$ . In what follows, we write  $Z_i$  (resp.,  $Z_i(\mathcal{S})$ ) instead of  $Z_i^W(\mathcal{S})$  when  $\mathcal{S}$  and  $W$  (resp., when  $W$ ) are clear from the context.

To reduce the search space for the possible hunter strategies, we establish that we can have winning hunter strategies that shoot only on a subset of contaminated vertices in each round, without increasing the number of hunters required. More precisely, a hunter strategy  $\mathcal{S} = (S_1, \dots, S_\ell)$  is said to be *parsimonious* if, for every  $1 \leq i \leq \ell$ ,  $S_i \subseteq Z_{i-1}(\mathcal{S})$ .

### 3 Monotonicity

In classical graph pursuit-evasion games, an important notion is that of *monotonicity*. On a high level, a strategy is *monotone* if the area reachable by the fugitive never increases. In the particular case of Graph Searching games, a strategy is monotone if, once a searcher is removed from some vertex, it is never necessary to occupy this vertex during a subsequent round. Monotone strategies have been widely studied [3, 29, 24] because it is generally easier to design them and because they have length polynomial in the size of the graph and so, corresponding decision problems can be proven to be in NP.

It is clear that such a definition is not suitable to the HUNTERS AND RABBIT game. Indeed, consider the graph that consists of a single edge  $uv$ : the hunter must shoot at some vertex, say  $u$ , and, if the rabbit was at  $v$ , it will move to  $u$ , i.e., the vertex  $u$  is “recontaminated”.



■ **Figure 1** Example of a bipartite graph (where  $V_r = \{a, c, e, g\}$  corresponds to the red part of the bipartition, illustrated by the red vertices in the figures) and of a parsimonious winning strategy with respect to  $V_r$ , such that no vertex in  $\{a, e, g\}$  is ever shot. Each subfigure depicts the situation at the end of the corresponding round. The cross indicates the shot of the hunter, and all the possible positions of the rabbit are shown by a rabbit next to the corresponding vertex.

Therefore, we propose to define monotonicity in the HUNTERS AND RABBIT game as follows (see the formal definition below): once a vertex has been “cleared”, if the rabbit can access it in a subsequent round, then the vertex must be shot immediately.

In classical Graph Searching games, a vertex being cleared at some round means that the searcher’s strategy ensures that the fugitive cannot occupy this vertex at this round. A recontaminated vertex can be intuitively defined as a vertex that can be reached by the fugitive while having been cleared in a previous round. This intuitive definition does not make any sense in the HUNTERS AND RABBIT game. For example, it is shown in [1] that, in bipartite graphs,  $h(G) = h_{V_r}(G) = h_{V_w}(G)$ , where  $(V_r, V_w)$  denotes the bipartition of  $V$ . In other words, it is sufficient to consider winning hunter strategies with respect to one of the independent sets of the bipartition; we call this the *red variant* of the game. In this case, every vertex of  $V_r$  is cleared at every odd round (since the rabbit can only occupy vertices of  $V_w$  at odd rounds) and so looking for a strategy without recontamination would be meaningless.

A related difficulty comes from the fact that, contrary to classical Graph Searching games, a vertex may be “cleared” without having been shot during the game. As a concrete example, consider a star with three leaves whose edges have been subdivided once each. Then, assuming that the leaves and the centre are red, in the red variant, it is possible for one hunter to win without shooting any of the leaves. Indeed, consider the strategy illustrated in Figure 1.

To overcome these difficulties, we propose to define the clearing of a vertex at some round by the fact that the actions of the hunters ensure that this vertex cannot be occupied by the rabbit at this round. Precisely, two actions of the hunters may clear a vertex: either a

hunter shoots a vertex  $v$  at round  $i$  and does not shoot the rabbit (meaning that  $v \neq r_{i-1}$ ), or the hunters shoot on every contaminated vertex in the neighbourhood of  $v$ . In this case, either  $v$  was occupied and the rabbit has to leave  $v$ , or  $v$  was not occupied and cannot be occupied after the move of the rabbit. In both cases,  $v \notin Z_i$ . This discussion motivates the following definition for the monotonicity of hunter strategies.

► **Definition 1** (Monotone strategy). *Let  $G$  be a graph and  $\mathcal{S}$  be a winning hunter strategy with respect to  $W \subseteq V(G)$ . We say that a vertex  $v$  is cleared at round  $i$  if either  $v \in S_i$  or,  $N(v) \cap Z_{i-1} \neq \emptyset$  and  $N(v) \cap Z_{i-1} \subseteq S_i$ . A strategy  $\mathcal{S} = (S_1, \dots, S_\ell)$  is monotone if, for every vertex  $v \in V$ , if there exists an  $i$  such that  $v$  is cleared at round  $i$ , then for every  $j > i$  such that  $v \in Z_j$ , the strategy ensures that  $v \in S_{j+1}$ . A vertex  $v$  is recontaminated at round  $j$  if there exists  $i \leq j$  such that  $v$  is cleared at round  $i$  and  $v \in Z_j \setminus S_{j+1}$ .*

The *monotone hunter number* of a graph  $G$  with respect to  $W \subseteq V(G)$ , denoted by  $mh_W(G)$ , is the minimum number  $k$  such that  $k$  hunters have a monotone winning hunter strategy in  $G$  with respect to  $W$ . Let us denote the *monotone hunter number*  $mh_V(G)$  of  $G$  by  $mh(G)$ . Note that, by definition,  $h_W(G) \leq mh_W(G) \leq mh(G)$ .

We can prove that monotone strategies have many interesting properties that are used in most of the proofs of our results. The proofs of these properties, which are omitted due to lack of space, are not trivial. The most intuitive one, is that, when the hunters follow a monotone strategy  $\mathcal{S}$ , the set of possible positions of the rabbit cannot increase. That is,  $Z_\ell(\mathcal{S}) \subseteq \dots \subseteq Z_1(\mathcal{S})$ . Moreover,  $mh(G)$  is closed under taking subgraphs. A crucial property is that there exists an optimal (using  $mh(G)$  hunter) parsimonious monotone strategy in any graph  $G$ . Finally, in any monotone strategy, once a vertex is shot, it has to be continuously shot until the rabbit can no longer reach it. In particular, this last property is used in the proof of upcoming Theorem 2 and it implies that the problem of computing  $mh$  is in NP.

### 3.1 Monotone hunter number and pathwidth

Here, we relate the monotone hunter number of a graph to its pathwidth. Our result might be surprising since the pathwidth of a graph  $G$  is equivalent to the number of searchers required to (monotonously) capture an arbitrary fast invisible fugitive [3] while, in our case, the invisible rabbit seems much weaker than the fugitive: the rabbit is “slow” (it moves only to neighbours) and constrained to move at every round. In this view, one might guess that the monotone hunter number of a graph could be arbitrary smaller than its pathwidth. On the contrary, we show that these parameters differ by at most one.

A *path-decomposition* of a graph  $G = (V, E)$  is a sequence  $P = (X_1, \dots, X_p)$  of subsets of vertices, called *bags*, such that (1)  $\bigcup_{i \leq p} X_i = V$ ; (2) for every  $uv \in E$ , there exists  $i \leq p$  with  $\{u, v\} \subseteq X_i$ ; and (3): for every  $1 \leq i < j \leq p$ ,  $X_i \cap X_j \subseteq X_{i-1} \cup X_{j+1}$ . The *width*  $w(P)$  of  $P$  is the size of a largest bag of  $P$  minus one, i.e.,  $w(P) = \max_{i \leq p} |X_i| - 1$ . The *pathwidth*  $pw(G)$  of  $G$  is the minimum width of its path-decompositions. A path-decomposition of  $G$  of width  $pw(G)$  is said to be *optimal*.

► **Theorem 2.** *For any graph  $G = (V, E)$ ,  $pw(G) \leq mh(G) \leq pw(G) + 1$ .*

**Sketch of proof.** Let  $P = (X_1, \dots, X_\ell)$  be a path-decomposition of  $G$  with width  $k$ . Then,  $P$  is a monotone hunter strategy in  $G$  using  $k + 1$  hunters.

To show the other inequality, let  $\mathcal{S} = (S_1, \dots, S_\ell)$  be a parsimonious winning monotone hunter strategy in  $G$  using at most  $k \geq mh(G)$  hunters. Observe that  $\mathcal{S}$  is almost a path-decomposition due to the fact that it is parsimonious and monotone. Indeed, any vertex  $v$  that is shot by  $\mathcal{S}$ , will be shot during some consecutive rounds (will belong to consecutive

bags of the decomposition). It remains to take care of unshot vertices. Such vertices are cleared by shooting at all of their neighbours during a round of  $\mathcal{S}$ . So, starting from  $\mathcal{S}$ , we build a path-decomposition  $\mathcal{P}$  of  $G$  as follows: we start with a bag  $B_i$  corresponding to  $S_i$ , for each  $i$ . Then, for each vertex  $u$  that has never been shot, all the neighbours of which are shot during the round  $i + 1$  for the first time, we create an intermediate bag, between  $B_i$  and  $B_{i+1}$ , containing all the vertices of  $B_i$  and the vertex  $u$ . ◀

Theorem 2 has important consequences, following from the inapproximability of the pathwidth of a graph [4]. Moreover, using a result in [17], this implies that recontamination may help in the HUNTERS AND RABBIT game.

► **Corollary 3.** *Given an  $n$ -node graph  $G$  and  $k \in \mathbb{N}$ , it is NP-complete to decide whether  $mh(G) \leq k$ . Moreover, it is NP-hard to approximate  $mh(G)$  up to an additive error of  $n^\varepsilon$ , for  $0 < \varepsilon < 1$ .*

► **Corollary 4.** *There exists  $\varepsilon > 0$  such that, for any  $k \in \mathbb{N}$ , there exists a tree  $T$  with  $h(T) \geq k$  and  $mh(T) \geq (1 + \varepsilon)h(T)$ .*

## 4 (Monotone) hunter number of some graph classes

In this section, we characterise the (monotone) hunter number of several graph classes such as cographs, split and interval graphs. In all these cases, our results lead to a polynomial time algorithm to compute the monotone hunter number.

### 4.1 Split and interval graphs

A graph  $G = (V, E)$  is a *split graph* if  $V = C \cup I$  can be partitioned into two sets  $C$  and  $I$ , inducing an inclusion-maximal clique and an independent set, respectively. Given a split graph  $G$ , a partition  $(C, I)$  of  $V(G)$  can be computed in linear time [18]. The following theorem fully characterises the hunter number of split graphs. It also allows us to show that the hunter number and the pathwidth of split graphs coincide.

► **Theorem 5.** *Let  $G = (C \cup I, E)$  be a split graph. Then,  $h(G) = |C|$  if and only if for every two distinct vertices  $x, y \in C$ , there exists a vertex  $z \in N_I(x) \cap N_I(y)$ . Otherwise,  $h(G) = |C| - 1$ .*

**Sketch of proof.** If every two vertices of  $C$  have a common neighbour in  $I$ , there exists a rabbit strategy against  $|C| - 1$  hunters. The idea is to take advantage of the fact that not all the vertices of  $C$  can be shot during a same round. Thus, the rabbit will remain as much as possible on  $C$ , and if it is unable to do so, it will go to  $I$  and then return to  $C$ , which is possible due to the hypothesis. For the reverse direction, let  $x$  and  $y$  be two vertices of  $C$  without common neighbour in  $I$ . The strategy  $\mathcal{S} = (S_1, S_2, S_3, S_4, S_5)$ , where  $S_1 = S_2 = S_5 = C \setminus \{y\}$  and  $S_3 = S_4 = C \setminus \{x\}$ , is a winning hunter strategy using  $|C| - 1$  hunters. ◀

Recall that a vertex in a graph  $G$  is *simplicial* if its neighbourhood induces a clique. Recall also that an *interval graph* is the intersection graph of a set of intervals in the real line. Let  $\omega(G)$  denote the maximum size of a clique of  $G$ .

► **Theorem 6.** *Let  $G$  be a interval (resp. split) graph. Then,  $h(G) = mh(G) = \omega(G) - 1$  (resp.  $mh(G) = \omega(G) - 1$ ) if every maximum clique has a simplicial vertex. Otherwise,  $mh(G) = \omega(G)$ .*

**Sketch of proof.** For the first direction, in case  $G = (C \cup I, E)$  is a split graph with  $v$  being a simplicial vertex of  $C$ ,  $S = (C \setminus v, C \setminus v)$  is a monotone winning hunter strategy using  $|C| - 1$  hunters. In case  $G$  is an interval graph, recall that  $pw(G) = \omega(G) - 1$  and  $G$  admits an optimal path-decomposition where each bag induces a complete graph. We adapt such an optimal path-decomposition, to a hunter strategy using  $\omega(G) - 1$  hunters, by removing a simplicial vertex from each bag containing a maximum clique and shooting twice at each of these bags.

For the reverse direction, let  $C$  be any maximum clique of  $G$  without simplicial vertex. Assume that there exists a monotone hunter strategy  $\mathcal{S}$  using  $|C| - 1$  hunters. Then there exists at least one round such that  $C$  hunters shoot on vertices of  $C$  (otherwise the rabbit can survive by staying on  $C$ ). Consider the first such round and let  $u$  be the vertex of  $C$  that is not shot. Using the fact that  $C$  has no simplicial vertex, we prove that there exists  $w \in N(u) \setminus C$  such that both  $u$  and  $w$  have never been shot until this round. Thus, the rabbit can oscillate between  $u$  and  $w$  either until the end, or until the first round where at least one of  $u$  or  $w$  is shot, where it recontaminates a vertex of  $C$ . This is a contradiction. ◀

From the above, we can show that there exist split and interval graphs  $G$  for which  $mh(G) \neq h(G)$ , i.e., recontamination helps in these graph families. Note also that, even knowing that  $\omega(G) - 1 \leq h(G) \leq \omega(G)$  for interval graphs, computing  $h(G)$  when some maximum clique has no simplicial vertex is challenging.

## 4.2 Cographs

The class of *cographs* can be defined recursively as follows [10]. One vertex is a cograph. Given two cographs  $A$  and  $B$ , their disjoint union  $A \cup B$  is a cograph, and their join  $A \bowtie B$  (where all edges between  $A$  and  $B$  are added) is a cograph. A decomposition of a cograph (i.e., a building sequence of unions and joins performed from single vertices) can be computed in linear time [10].

► **Theorem 7.**  *$mh(G)$  can be computed in linear time in the class of cographs.*

**Sketch of proof.** It suffices to calculate  $mh(G)$  in the case where  $G = A \bowtie B$ . It is easy to see that  $mh(A \bowtie B) \leq m = \min(mh(A) + |V(B)|, |V(A)| + mh(B))$ . Assume that there exists a winning monotone hunter strategy  $\mathcal{S} = (S_1, \dots, S_\ell)$  using at most  $m - 1$  hunters. Let  $v$  be the first vertex that is no longer available for the rabbit and assume, w.l.o.g., that  $v \in V(A)$ . Then, all the vertices in  $N_B(v) = V(B)$  were shot. Thus,  $V(B)$  must be shot during every subsequent round. This means that  $\mathcal{S}$  can clear  $A$  using  $m - |V(B)| - 1 < mh(A)$  hunters, a contradiction. ◀

Once again, the case of the hunter number seems more challenging. In particular, the following lemma shows that recontamination may help in cographs.

► **Lemma 8.** *For every  $k \geq 1$ , there exists a cograph  $G$  such that  $h(G) \geq k$  and  $mh(G) \geq \frac{3}{2}h(G) - 1$ .*

**Sketch of proof.** Let  $A$  and  $B$  be two (isomorphic) cographs consisting of the disjoint union of a complete graph with  $a \geq 3$  vertices and  $a$  independent vertices. The graph  $G = A \bowtie B$  verifies that  $mh(G) = 3a - 1$  and  $h(G) = 2a$ . ◀

## 5 (Monotone) hunter number of trees

This section is devoted to showing that  $mh$  can be computed in polynomial-time in the class of trees. Then, we show that recontamination helps a lot in trees.

### 5.1 Monotone hunter number in trees

Let  $T$  be a tree and  $v \in V(T)$ . A *branch* at  $v$  is any connected component of  $T - v$ . A *star* is any tree with at least two vertices and at most one vertex with degree at least three. Roughly, Parsons' Lemma [26] states that, for any tree  $T$  and  $k \in \mathbb{N}$ ,  $pw(T) \geq k + 1$  if and only if there exists a vertex  $v$  such that at least three branches at  $v$  have pathwidth at least  $k$ . Here, we adapt this lemma in the case of the monotone hunter number of trees.

► **Lemma 9** (Parsons' like lemma). *Let  $T = (V, E)$  be any tree.*

- $mh(T) = 0$  if and only if  $|V| = 1$ ;
- $mh(T) = 1$  if and only if  $T$  is a star;
- $mh(T) = 2$  if and only if  $T$  is not a star and contains a path  $P$  such that  $T \setminus P$  is a forest of stars and isolated vertices;
- For every  $k \geq 3$ ,  $mh(T) \geq k$  if and only if there exists a vertex  $v \in V$  such that at least three branches at  $v$  have monotone hunter number at least  $k - 1$ .

Taking advantage of Lemma 9, we design a dynamic programming algorithm to compute  $mh(T)$  of a given tree  $T$ . Our algorithm is heavily inspired by the polynomial time algorithm computing the pathwidth of  $T$  [13].

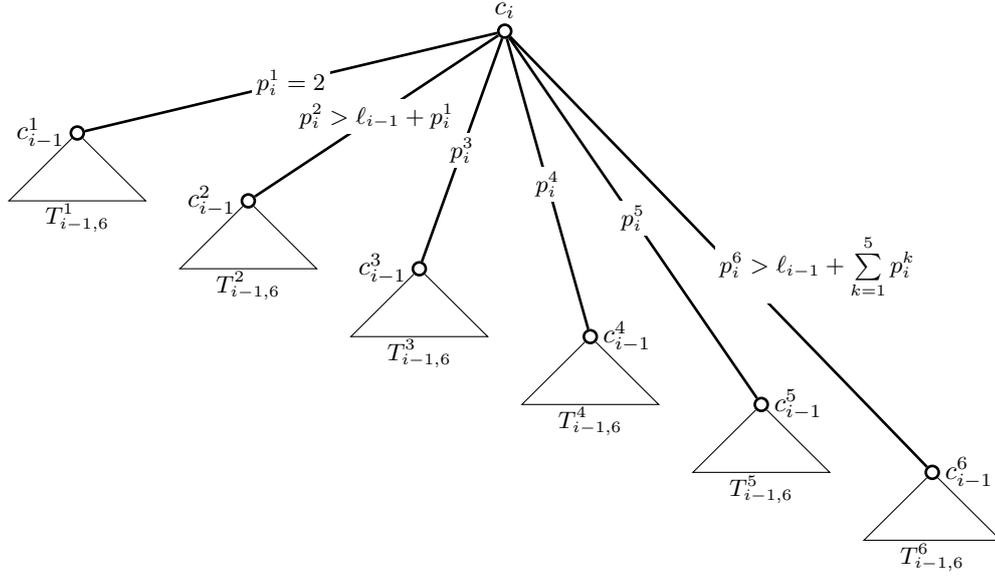
► **Theorem 10.** *For a tree  $T$ ,  $mh(T)$  can be computed in polynomial time.*

### 5.2 Monotone hunter number in the red variant in trees

So far, we have investigated the monotone HUNTERS AND RABBIT, since monotone strategies are often easier to deal with. Previous works on the HUNTERS AND RABBIT game in bipartite graphs  $G = (V_r \cup V_w, E)$  have shown that studying the *red variant* of the HUNTERS AND RABBIT game, i.e., when the rabbit is constrained to start in a vertex in  $V_r$ , could be very fruitful. For instance,  $h(G) = h_{V_r}(G)$  holds for every bipartite graph  $G = (V_r \cup V_w, E)$ , which helped to get many results on the HUNTERS AND RABBIT game [1, 5, 17]. Therefore, it is interesting to consider the monotonicity constraint when restricted to the red variant of the HUNTERS AND RABBIT game. We now focus on investigating the difference between  $h_{V_r}(T)$  and  $mh_{V_r}(T)$ , for any tree  $T$ . Thankfully, all the useful properties we observed for the monotone version can be adapted for the monotone red variant as well.

By Corollary 4, there exists  $\varepsilon > 0$  such that, for any  $k \in \mathbb{N}$ , there exists a tree  $T$  with  $h(T) \geq k$  and  $mh(T) \geq (1 + \varepsilon)h(T)$ . In the following theorem we improve this, by showing that there exists an infinite family of trees  $T$  such that the difference between  $mh(T)$  and  $h(T)$  is arbitrarily large. This follows from the next theorem since  $h_{V_r}(T) = h(T)$  and  $mh(T) \geq mh_{V_r}(T)$  for any tree  $T$ . Its proof follows from Lemmas 12 and 13, presented below.

► **Theorem 11.** *For every  $i \geq 3$ , there exists a tree  $T$  such that  $mh_{V_r}(T) \geq i$  and  $h_{V_r}(T) = 2$ .*



■ **Figure 2** The graph  $T_{i,6}$ . The labels on the edges are used to represent their respective lengths. In particular, for every  $2 \leq j \leq 6$ , we have that  $p_i^j > \sum_{1 \leq k \leq j-1} p_i^k + \ell_{i-1}$ , where  $p_i^1 = 2$  and  $\ell_{i-1}$  is equal to the number of turns needed to clear any copy of the  $T_{i-1,6}$  graph.

**The construction of the tree  $T_{i,q}$ .** Let  $S_{k,q}$  be the rooted tree obtained from  $q \geq 6$  paths of length  $k \geq 3$  (with  $k$  edges) by identifying an endpoint of each path into a common vertex called the *root* of  $S_{k,q}$  and denoted by  $c$ . From now on, let  $(V_r, V_w)$  be the bipartition of  $V(S_{k,q})$  and assume that  $c \in V_r$ . Let  $\mathcal{S}_1$  be a winning hunter strategy such that  $mh_{V_r}(S_{3,q}) = 2$  [7], and let  $\ell_1$  be the smallest even integer greater or equal to the length of  $\mathcal{S}_1$ .

For every  $i \geq 2$  and  $q \geq 6$ , let  $T_{i,q}$  be the tree recursively built as follows (see Figure 2). First,  $T_{1,q} = S_{3,q}$ . Then, for  $i > 1$ , we assume that  $T_{i-1,q}$  has been defined recursively and that there exists a winning hunter strategy, of length  $\ell_{i-1}$ , using 2 hunters in the red variant in  $T_{i-1,q}$ .  $T_{i,q}$  is obtained from  $q$  vertex disjoint copies  $T_i^1, \dots, T_i^q$  of  $T_{i-1,q}$  and from a vertex  $c_i$ , the root of  $T_{i,q}$ . Then, for every  $1 \leq j \leq q$ , add a path  $P_i^j$  of length  $p_i^j$  between the root  $c_i^j$  of  $T_i^j$  and  $c_i$  (that is,  $c_i$  and  $c_i^j$  are at distance  $p_i^j$  in  $T_{i,q}$ ). The lengths  $p_i^j$  are defined recursively as follows. Let  $p_i^1 = 2$  and, for every  $1 < j \leq q$ , let  $p_i^j$  be the minimum even integer greater or equal to  $\ell_{i-1} + \sum_{1 \leq k < j} p_i^k$ . Finally, let us assume that  $c_i \in V_r$  and note that, since  $p_i^j$  is even, this implies that  $c_i, c_i^1, \dots, c_i^q$  all belong to  $V_r$ .

► **Lemma 12.** For any  $i \in \mathbb{N}^*$  and  $q \geq 6$ ,  $h_{V_r}(T_{i,q}) = 2$ .

**Sketch of proof.** It suffices to give a winning hunter strategy using 2 hunters. The strategy  $\mathcal{S}^i$  is defined recursively, and consists of  $q$  phases. During the  $j^{\text{th}}$  phase, for  $1 \leq j \leq q$ , one hunter shoots at  $c_i$ , while the other “pushes” the rabbit toward the subtrees  $T_i^q$ , then  $T_i^{q-1}$ , and so on, until  $T_i^j$ . Then, the two hunters clear the subtree  $T_i^j$ , following the strategy  $\mathcal{S}^{i-1}$  (without the rabbit being able to leave  $T_i^j$  if it was there). The lengths of the paths linking the roots of the subtrees to  $c_i$  guarantee that the rabbit cannot reach  $c_i$  before  $T_i^j$  has been cleared. ◀

As a consequence, the hunter number is not closed under taking a minor.

► **Lemma 13.** For  $i \geq 3$  and  $q \geq 2i$ ,  $mh_{V_r}(T_{2i,q}) \geq i$ .



► **Theorem 14.** *Let  $G$  be a graph having a vertex cover  $U$  of size  $t$ , and let  $k \in \mathbb{N}$ . Deciding whether  $h(G) \leq k$  (resp.,  $mh(G) \leq k$ ) is FPT parameterized by  $t$ . More specifically, these problems admit a kernel with at most  $4^t(t+1) + 2t$  vertices.*

**Sketch of proof.** First, observe that  $mh(G) \leq t$  since the hunters have a monotone winning strategy by shooting the vertices of  $U$  twice. Second, we argue that the standard rule of removing twins from  $I$  leads to an exponential kernel for these questions. More specifically, we have the following reduction rule along with the rule that if  $k \leq t$ , return a Yes-instance: If there is some  $I' \subset I$  such that  $|I'| > k+1$  and for any two vertices  $x, y \in I'$ ,  $N(x) = N(y)$ , then delete an arbitrary vertex, say  $v$ , from  $I'$  to get  $G'$  and let  $k' = k$ . It is not difficult to see that this will give us a kernel with at most  $2^t(t+1) + t$  vertices.

Next, we give an intuition regarding the safeness of our reduction rule. It is easy to see that if  $(G, k)$  is a Yes-instance, then  $(G', k)$  is also a Yes-instance as both  $h(G)$  and  $mh(G)$  are closed under taking subgraphs. For the reverse direction, and towards a contradiction, we will assume that  $(G, k)$  is a no-instance but  $(G', k)$  is a yes-instance. Since  $(G', k)$  is a yes instance, there exists a winning hunter strategy  $\mathcal{S}'$  using at most  $k$  hunters in  $G'$ . Since  $(G, k)$  is a no-instance, there exists a rabbit strategy  $R$  winning against  $\mathcal{S}'$  in  $G$ . From  $R$ , we can design  $R'$ , an equivalent rabbit strategy winning against  $\mathcal{S}'$  in  $G'$ , contradicting that  $(G', k)$  was a yes-instance. ◀

## 7    **Some Future Directions**

In this paper, we studied the HUNTERS AND RABBIT game by defining the notion of monotonicity for this game. Using this notion of monotonicity, we characterised the monotone hunter number for various classes of graphs. Moreover, we established that, unlike several Graph Searching games, the monotonicity helps in this game, i.e.,  $h(G)$  can be arbitrary smaller than  $mh(G)$ .

There are still several challenging open questions in this area. The most important among them is the computational complexity of HUNTERS AND RABBIT. Although our results establish that computing  $mh(G)$  is NP-hard, the computational complexity of computing/deciding  $h(G)$  remains open, even if  $G$  is restricted to be a tree.

We also established that both HUNTERS AND RABBIT, as well as its monotone variant, are FPT parameterised by  $vc(G)$  by designing exponential kernels. It is not difficult to see that both of these variants admit AND Composition parameterised by the solution size (by taking the disjoint union of the instances). Thus, since computing  $mh(G)$  is NP-hard and  $pw(G) \leq mh(G) \leq pw(G) + 1$ , it is unlikely for MONOTONE HUNTERS AND RABBIT parameterized by  $k + pw(G)$  to admit a polynomial compression. Note that the same cannot be argued about HUNTERS AND RABBIT since it is not yet proved to be NP-hard. Moreover, since  $mh(G)$  is closely related to  $pw(G)$  and pathwidth admits a polynomial kernel with respect to  $vc(G)$  [8], it might be interesting to see if deciding  $mh(G) \leq k$  (resp.,  $h(G) \leq k$ ) also admits a polynomial kernel when parameterised by  $vc(G)$ . Moreover, another interesting research direction is to study the parameterised complexity of both these games by considering parameters such as solution size, treewidth, and pathwidth.

Finally, we propose some open questions concerning the computation of  $h(G)$  for various graph classes including trees, cographs, and interval graphs. Specifically, it will be interesting to design a polynomial time algorithm to compute  $h(T)$  for a tree  $T$ , a challenge that was already proposed in [1]. The natural way that one could tackle this question is through the notion of monotonicity, which we defined and studied in this paper. Unfortunately, Theorem 11 implies that such an approach will not work. This means that a positive answer to this question (if any) would require the introduction of new tools and techniques. Moreover, it would be interesting to know the monotone hunter number of grids.

---

**References**

---

- 1 T. V. Abramovskaya, F. V. Fomin, P. A. Golovach, and M. Pilipczuk. How to hunt an invisible rabbit on a graph. *European Journal of Combinatorics*, 52:12–26, 2016.
- 2 I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, 2004.
- 3 D. Bienstock and P. D. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.
- 4 H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.
- 5 J. Bolkema and C. Groothuis. Hunting rabbits on the hypercube. *Discrete Mathematics*, 342(2):360–372, 2019.
- 6 R. L. Breisch. An intuitive approach to speleotopology. *Southwestern cavers*, 6(5):72–78, 1967.
- 7 J. R. Britnell and M. Wildon. Finding a princess in a palace: a pursuit-evasion problem. *The Electronic Journal of Combinatorics*, 20(1):25, 2013.
- 8 M. Chapelle, M. Liedloff, I. Todinca, and Y. Villanger. Treewidth and pathwidth parameterized by the vertex cover number. *Discrete Applied Mathematics*, 216:114–129, 2017.
- 9 T. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous Robots*, 31(299):299–316, 2011.
- 10 D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
- 11 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.
- 12 T. Dissaux, F. Fioravantes, H. Gahlawat, and N. Nisse. Further results on the Hunters and Rabbit game through monotonicity. Technical report, Inria - Sophia Antipolis, February 2023. URL: <https://hal.science/hal-03995642>.
- 13 J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.
- 14 H. Gahlawat and M. Zehavi. Parameterized analysis of the cops and robber game. In B erome Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 47:1–40:15, Dagstuhl, Germany, 2023.
- 15 A. C. Giannopoulou, P. Hunter, and D. M. Thilikos. Lifo-search: A min-max theorem and a searching game for cycle-rank and treedepth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.
- 16 G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, 2003.
- 17 V. Gruslys and A. M eroueh. Catching a mouse on a tree. *arXiv preprint*, 2015. arXiv:1502.06591.
- 18 P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1:275–284, 1981.
- 19 J. Haslegrave. An evasion game on a graph. *Discrete Mathematics*, 314:1–5, 2014.
- 20 D. Ilcinkas, N. Nisse, and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22(2):117–127, 2009.
- 21 A. Isaza, J. Lu, V. Bulitko, and R. Greiner. A cover-based approach to multi-agent moving target pursuit. In *proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 54–59. AAAI Press, 2008.
- 22 T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory Series B*, 82(1):138–154, 2001.
- 23 A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.

## 42:14 Recontamination Helps a Lot to Hunt a Rabbit

- 24 F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. *Theoretical Computer Science*, 399(3):169–178, 2008.
- 25 N. Nisse. Network decontamination. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 516–548. Springer, 2019.
- 26 T. D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs, LNCS*, volume 642, pages 426–441. Springer, 1978.
- 27 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.
- 28 D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: [http://www.cambridge.org/de/knowledge/isbn/item5759340/?site\\_locale=de\\_DE](http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE).
- 29 B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.

# String Diagrammatic Trace Theory

Matthew Earnshaw   

Department of Software Science, Tallinn University of Technology, Estonia

Paweł Sobociński   

Department of Software Science, Tallinn University of Technology, Estonia

---

## Abstract

---

We extend the theory of formal languages in monoidal categories to the multi-sorted, symmetric case, and show how this theory permits a graphical treatment of topics in concurrency. In particular, we show that Mazurkiewicz trace languages are precisely *symmetric monoidal languages* over *monoidal distributed alphabets*. We introduce *symmetric monoidal automata*, which define the class of regular symmetric monoidal languages. Furthermore, we prove that Zielonka’s asynchronous automata coincide with symmetric monoidal automata over monoidal distributed alphabets. Finally, we apply the string diagrams for symmetric premonoidal categories to derive serializations of traces.

**2012 ACM Subject Classification** Theory of computation → Concurrency; Theory of computation → Formal languages and automata theory; Theory of computation → Categorical semantics

**Keywords and phrases** symmetric monoidal categories, Mazurkiewicz traces, asynchronous automata

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.43

**Funding** Estonian Research Council grant PRG1210 and the European Union under Grant Agreement No. 101087529. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

**Acknowledgements** We thank Chad Nester, Mario Román, and Niels Voorneveld for discussions.

## 1 Introduction

*Monoidal languages* [12] generalize formal languages of words to formal languages of *string diagrams*. String diagrams [16, 29] are a graphical representation of morphisms in *monoidal categories*. Monoidal categories can be considered *2-dimensional monoids* [6]: just as monoids are categories with one object, whose morphisms are elements of the monoid, (strict) monoidal categories can be defined as 2-categories with one object. Accordingly, *monoidal languages* are subsets of morphisms in free monoidal categories, just as word languages are subsets of free monoids. *Regular* monoidal languages are those specifiable by finitary grammars or automata. Our paper [12] introduced these devices and examined properties of languages in single-sorted, planar monoidal categories. These include regular languages of words and trees, but also languages of *planar* string diagrams that are neither linear nor tree-like.

In this paper, motivated by concurrency theory, we extend this theory to *coloured props*: multi-sorted monoidal categories with symmetries (Section 2). The resulting theory of *symmetric* monoidal languages (Section 3) captures languages of diagrams having multiple colours of string and in which strings may cross, permitting non-planar diagrams. In terms of concurrency, colours represent different *runtimes*, or *threads of execution*.

Indeed, in Section 4 we show that Mazurkiewicz trace languages [21] are exactly symmetric monoidal languages over alphabets of a particular shape called *monoidal distributed alphabets*. In Section 5 we introduce automata for symmetric monoidal languages, defining the class of *regular* symmetric monoidal languages. Then, in Section 6 we show that these are exactly the asynchronous automata of Zielonka [32] when instantiated over monoidal distributed alphabets. Finally, in Section 7 we use the algebra of symmetric *premonoidal* categories to show how serialization of traces can be treated string-diagrammatically.



© Matthew Earnshaw and Paweł Sobociński;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 43; pp. 43:1–43:15

Leibniz International Proceedings in Informatics



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

**Related work**

Our previous work [12] introduced monoidal languages in the planar, single-sorted case; that is, languages of morphisms in free *props*. Similar languages of graphs were studied by Bossut [5], but their underlying algebra was not made explicit. Here, we again leverage the algebraic perspective, extending our theory to symmetric multi-sorted monoidal categories (props).

In the introduction to Joyal & Street’s foundational work on string diagrams for monoidal categories [16], it is suggested that string diagrams have a connection to the *heaps* of Viennot [30]. Heaps are known to be equivalent to Mazurkiewicz trace monoids (also known as partially commutative monoids) [17], but a precise formulation of the suggested relation with string diagrams has not appeared in the literature until now.

The notion of dependence graph [13] has also been used to give a topological presentation of Mazurkiewicz traces. Our use of the algebra of monoidal categories, rather than graphs, has various advantages. For example, we can apply our language theory for monoidal categories to traces, and we see notions such as asynchronous automata arise naturally from this. It also suggests generalizations of trace languages, in particular going beyond the case of atomic actions (Remark 23). Finally, it brings our work into proximity with the semantics of Petri nets and other formalisms for concurrency based on monoidal categories [2, 24].

**2 Monoidal Graphs, Props and their String Diagrams**

In this section we recall the basic definitions used in the following, including the specific flavour of monoidal categories known as *props* [20], along with their string diagrams [16, 29]. Just as a category can be presented by a directed graph, (strict) monoidal categories can be presented by *monoidal graphs*, a kind of multi-input, multi-output directed graph.

► **Definition 1.** A monoidal graph  $\mathcal{G}$  is a set  $B_{\mathcal{G}}$  of boxes, a set  $S_{\mathcal{G}}$  of sorts, and functions  $s, t : B_{\mathcal{G}} \rightrightarrows S_{\mathcal{G}}^*$  to the free monoid over  $S_{\mathcal{G}}$ , giving source and target boundaries of each box.

The alphabets of monoidal languages will be finite monoidal graphs: those in which  $B_{\mathcal{G}}$  and  $S_{\mathcal{G}}$  are both finite sets. In fact, since we are interested in finite state machines over finite alphabets, we will work exclusively with finite monoidal graphs. Diagrammatically, a (finite) monoidal graph can be pictured as a collection of boxes, labelled by elements of  $B_{\mathcal{G}}$  with strings entering on the left and exiting on the right, labelled by sorts given by the source and target functions. For example, the following depicts the monoidal graph  $\mathcal{G}$  with  $B_{\mathcal{G}} = \{\gamma, \gamma'\}$ ,  $S_{\mathcal{G}} = \{A, B\}$ ,  $s(\gamma) = AB$ ,  $t(\gamma) = ABA$ ,  $s(\gamma') = A$ ,  $t(\gamma') = BB$ :

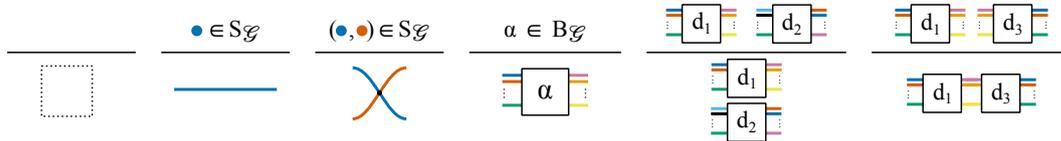


Sorts of a monoidal graph are sometimes called *colours*, since we could equally use different colours of string to represent different sorts, and we shall do so in places below. For a box  $\gamma \in B_{\mathcal{G}}$  we call  $s(\gamma)$  and  $t(\gamma)$  the *arity* and *coarity* of  $\gamma$ , respectively, and write  $\gamma : s(\gamma) \rightarrow t(\gamma)$ . We will also call  $\gamma$  considered together with its arity and coarity a *generator*.

Monoidal graphs are generating data for monoidal categories. Recall that a *strict monoidal category* is a category  $\mathcal{C}$ , equipped with a functor  $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$  (the *monoidal product*) and a unit object  $I \in \mathcal{C}$ , such that  $\otimes$  is associative and unital. A strict monoidal category is *symmetric* if there is a natural family of *symmetry morphisms*  $\sigma_{A,B} : A \otimes B \rightarrow B \otimes A$ , for each pair of sorts, satisfying  $\sigma_{B,A} \circ \sigma_{A,B} = 1_{A \otimes B}$ . The monoidal product turns the sets of objects and morphisms in  $\mathcal{C}$  into monoids. A *prop* is a symmetric strict monoidal category

whose monoid of objects is a free monoid.<sup>1</sup> While the above data can be intimidating to the non-expert, the free prop  $\mathcal{F}_\times \mathcal{G}$  on a monoidal graph  $\mathcal{G}$  can be described in an intuitive and straightforward way: its arrows are the *string diagrams* generated by  $\mathcal{G}$ .

► **Definition 2.** *The free prop  $\mathcal{F}_\times \mathcal{G}$  on a monoidal graph  $\mathcal{G}$  has monoid of objects  $S_\mathcal{G}^*$  and morphisms string diagrams inductively defined as follows: Left to right: the empty diagram*



is a diagram; for every sort, the string on that sort is a diagram; for every pair of sorts, the symmetric braiding is a diagram; the diagram for every generator  $\alpha$  is a diagram; for any two diagrams their vertical juxtaposition is a diagram; and for any two diagrams with matching right and left boundaries, the diagram obtained by joining the matching wires is a diagram (their composition). The monoidal product is given on objects by concatenation, on diagrams by juxtaposition, and the unit is the empty word.

The idea is simple: we treat generators like circuit components, and we have a supply of wires (identity morphisms). We also have the ability to cross wires, without tangling them; we do not distinguish over-crossings from under-crossings. A string diagram is then just any (open) circuit that we can build. This notation is sound and complete: an equation between morphisms of strict monoidal categories follows from their axioms if and only if it holds between string diagrams up to planar isotopy [16]. Working with string diagrams rather than the usual term syntax for morphisms is more intuitive, and leads to shorter proofs as the structural equations hold automatically: for example, interchange of morphisms (Figure 1, left), unbraiding of symmetries (centre), and sliding of morphisms past symmetries (right).



■ **Figure 1** These pairs of string diagrams are equal, reflecting the functoriality of  $\otimes$  (*interchange*), inverses of symmetries, and naturality of symmetries, respectively.

► **Definition 3.** *A morphism of monoidal graphs  $\varphi : \mathcal{H} \rightarrow \mathcal{G}$  is given by functions  $B_\varphi : B_\mathcal{H} \rightarrow B_\mathcal{G}$  and  $S_\varphi : S_\mathcal{H} \rightarrow S_\mathcal{G}$  compatible with source and target functions:  $S_\varphi^* \circ s = s \circ B_\varphi$  and  $S_\varphi^* \circ t = t \circ B_\varphi$ , where  $S_\varphi^*$  is the unique monoid homomorphism determined by  $S_\varphi$ .*

Morphisms of monoidal graphs freely generate morphisms of props: strict monoidal functors preserving sorts. Every prop has an underlying monoidal graph whose boxes are all the morphisms of the prop. This extends to an adjunction  $\mathcal{F}_\times \dashv \mathcal{U}$  between the categories of monoidal graphs and props, where  $\mathcal{U}$  takes the underlying monoidal graph of a prop [16].

Monoidal categories have been applied to the study of both computing and physical processes [8, 9, 18, 25]. In these contexts, the monoidal product represents parallel composition of processes, and interchange reflects the *independence* of processes running in parallel. This is the main feature of monoidal categories that we will leverage in our representation of *traces* (Section 4). The use of multi-sorted props will allow fine-grained control of interchange.

<sup>1</sup> Some literature takes prop to mean that the monoid of objects is generated by a single object (and so isomorphic to  $\mathbb{N}$ ), using the term *coloured prop* for the general case above.

### 3 Symmetric Monoidal Languages

Our paper [12] treated the case of languages, grammars and automata over single-sorted *pros* (strict monoidal categories *without* symmetries), corresponding to languages of *planar* string diagrams with one string colour. In this section we introduce the *multi-sorted* (or “coloured”) *symmetric monoidal languages*, which will be needed in the following to extend monoidal language theory to trace theory. In Section 5 we introduce the corresponding automata.

Just as a classical formal language is a subset of a free monoid, a symmetric monoidal language is a subset of morphisms in a free prop:

► **Definition 4.** *Let  $\Gamma$  be a finite monoidal graph. A symmetric monoidal language over  $\Gamma$  is a set of morphisms in the free prop  $\mathcal{F}_\times \Gamma$  over  $\Gamma$ .*

A morphism of finite directed graphs  $G \rightarrow \Sigma$ , where  $\Sigma$  is a graph with one vertex, amounts to a labelling of the edges of  $G$  by edges of  $\Sigma$ . This is the starting point of Walters’ definition of regular grammar [31], which inspires the following definition:

► **Definition 5.** *A regular monoidal grammar is a morphism of finite monoidal graphs.*

For a regular monoidal grammar  $M \xrightarrow{\varphi} \Gamma$ , the monoidal graph  $\Gamma$  is the *alphabet*, and the generators of  $M$ , with their labelling by  $\varphi$ , correspond to production rules: see Example 7.

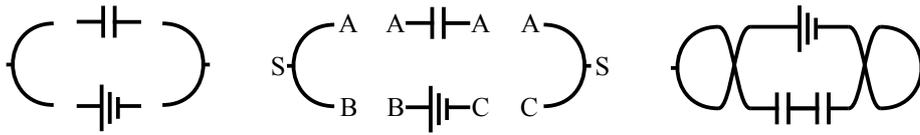
In the classical setting of word languages, a morphism of finite directed *graphs*  $G \rightarrow \Sigma$  determines a *regular language* over  $\Sigma$  once we specify initial and final state vertices in  $G$ . In a regular monoidal grammar  $M \rightarrow \Gamma$ , the “vertices” of  $M$  are words over  $S_M$ , leading to various natural choices of boundary condition (Remark 8). In this paper, we will take initial and final words over  $S_M^*$ . Specifying these words defines the symmetric monoidal language of the grammar (Definition 9), and we define the languages arising in this way to be the regular symmetric monoidal languages.

We illustrate these definitions with some pedagogical toy examples. In the remaining sections of this paper, we turn to our extended application in concurrency, and we shall see that Mazurkiewicz trace languages are a natural example of symmetric monoidal languages.

► **Example 6.** Let  $\varphi : M \rightarrow \Gamma$  be the regular monoidal grammar where  $M$  and  $\Gamma$  have a single sort ( $\bullet$ ) and no boxes, with  $S_\varphi(\bullet) = \bullet$ , and initial and final states  $n \in S_M^* \cong \mathbb{N}$ . Then the symmetric monoidal language of this grammar is the set of *permutations* of  $n$  wires: morphisms consisting only of symmetries and identities.

Props have been used to give syntax and semantics for various kinds of *signal flow graph* and *circuit diagrams* [1, 3, 4]. Intuitively, props are well suited for this purpose since wires may freely cross in a circuit.

► **Example 7.** We give a regular monoidal grammar for the syntax of (open) circuits with  $n \geq 0$  capacitors in series with a single voltage source (Figure 2). The alphabet  $\Gamma$  has a single sort, and boxes four circuit components (Figure 2, left). The monoidal graph  $M$  has four sorts  $\{S, A, B, C\}$  and four boxes  $s : S \rightarrow AB, c : A \rightarrow A, v : B \rightarrow C, s' : AC \rightarrow S$ .  $S_\varphi$  maps the four sorts to the single sort of  $\Gamma$ , and  $B_\varphi$  maps each box to a circuit component. We can draw the grammar  $\varphi : M \rightarrow \Gamma$  in a single diagram by drawing the graph for  $M$  but replacing each box  $b$  with its image under the grammar morphism  $B_\varphi(b)$  (Figure 2, centre). The initial and final languages are the single state  $\{S\}$ . Intuitively, the symmetric monoidal language determined by the grammar is all of the string diagrams  $S \rightarrow S$  that can be built using the “sorted” boxes of  $\Gamma$ , then forgetting the sorts.



■ **Figure 2** (Left) The alphabet  $\Gamma$ , giving syntax for circuits. (Centre) A regular monoidal grammar over  $\Gamma$ . (Right) An element of the regular symmetric monoidal language determined by this grammar.

► **Remark 8.** As mentioned above, there are various possible choices for the “initial and final states” of a monoidal grammar. In our previous paper [12], we took the empty word, giving languages of scalar string diagrams (i.e. no “dangling wires”): this neatly generalizes tree grammars. More generally, one can take initial and final *regular languages* of states over  $S_M$ , as considered by Bossut [5].

The free prop construction can be used to concisely describe the symmetric monoidal language of a regular monoidal grammar, defining the class of *regular symmetric monoidal languages*:

► **Definition 9.** Let  $(\varphi : M \rightarrow \Gamma, I, F)$  be a regular monoidal grammar equipped with regular languages  $I, F \subseteq S_M^*$ . This determines a symmetric monoidal language by taking the image of the set of morphisms  $\bigcup_{i \in I, f \in F} \mathcal{F}_\times M(i, f)$  under  $\mathcal{F}_\times \varphi$ , giving a set of morphisms in  $\mathcal{F}_\times \Gamma$ . The languages arising in this way are defined to be the regular symmetric monoidal languages.

In this paper, we will only need the case where  $I, F$  consist of single words. The slogan for the general case is that 2-dimensional regular languages have 1-dimensional regular boundaries. In Section 5, we will see that regular symmetric monoidal languages may equivalently be specified by *non-deterministic monoidal automata*.

► **Remark 10.** A regular monoidal grammar determines not only a regular symmetric monoidal language, but also a language in any algebraic structure generated by monoidal graphs, including planar monoidal categories (treated in [12]), and premonoidal categories (which we will use in Section 7). This is analogous to the way in which a finite labelled directed graph may generate both a subset of a free monoid, but also a subset of a free group, by freely adding inverses to the graph. Moreover, many properties of planar regular monoidal languages such as their closure properties proved in [12] only use grammars, and hence the same proofs work for languages in these other algebras.

## 4 Mazurkiewicz Trace Languages as Symmetric Monoidal Languages

The theory of Mazurkiewicz traces [10, 21, 23] provides a simple but powerful model of concurrent systems. Traces are a generalization of *words* in which specified pairs of letters can commute. If we think of letters as corresponding to atomic *actions*, then commuting letters reflect the *independence* of those particular actions and so their possible concurrent execution:  $ab$  is observationally indistinguishable from  $ba$  if  $a$  and  $b$  are independent.

In this section, we show that trace languages are symmetric monoidal languages over monoidal graphs of a particular form that we call *monoidal distributed alphabets*. In Section 5 we introduce *symmetric monoidal automata*, which operationally characterize the regular symmetric monoidal languages. In Section 6 we turn to *asynchronous automata* [32], a well-known model accepting exactly the *recognizable* trace languages, and show that these automata are precisely symmetric monoidal automata over monoidal distributed alphabets.

## 4.1 Independence and distribution

We recall some definitions from Mazurkiewicz trace theory, before recasting them in terms of monoidal languages. Fix a finite set  $\Sigma$ , an alphabet thought of as a set of atomic actions.

► **Definition 11.** An independence relation on  $\Sigma$  is a symmetric, irreflexive relation  $I$ . The induced dependence relation,  $D_I$  is the complement of  $I$ .

► **Definition 12.** For  $I$  an independence relation, let  $\equiv_I$  be the least congruence on  $\Sigma^*$  such that  $\forall a, b: (a, b) \in I \implies ab \equiv_I ba$ . The quotient monoid  $\mathcal{T}(\Sigma, I) := \Sigma^*/\equiv_I$  is the trace monoid.

► **Definition 13.** A (Mazurkiewicz) trace language over  $(\Sigma, I)$  is a subset of the trace monoid  $\mathcal{T}(\Sigma, I)$ .

An element of  $\mathcal{T}(\Sigma, I)$  or *trace over*  $(\Sigma, I)$  is thus an equivalence class of words up to commutation of independent letters. A trace language may be thought of as the set of possible observations of a concurrent system's behaviour, in which independent letters stand for actions which may occur concurrently. Independence relations correspond to *distributions*:

► **Definition 14** ([23]). A distribution of an alphabet  $\Sigma$  is a finite tuple of non-empty alphabets  $(\Sigma_1, \dots, \Sigma_k)$  such that  $\bigcup_{i=1}^k \Sigma_i = \Sigma$ .

► **Proposition 15** ([23]). A distribution of  $\Sigma$  corresponds to a function  $\text{loc} : \Sigma \rightarrow \mathcal{P}^+(\{1, \dots, k\}) : \sigma \mapsto \{i \mid \sigma \in \Sigma_i\}$ .

Such a function gives the set of “locations” of each action  $\sigma \in \Sigma$ . In terms of concurrency, we can consider this to be a set of memory locations, threads of execution, or runtimes in which  $\sigma$  participates. In particular, every action has a non-empty set of locations.

A well-known construction [23] allows us to move between independence relations and distributions: locations correspond to maximal cliques in the graph of the dependency relation. We recall this construction in the proof of Proposition 16, which refines this correspondence.

Let  $\text{Ind}_\Sigma$  be the poset of independence relations on  $\Sigma$ , with order the inclusion of relations. Similarly, define a preorder  $\text{Dist}_\Sigma$  on distributions by  $(\Sigma_1, \dots, \Sigma_p) \leq (\Sigma'_1, \dots, \Sigma'_q)$  iff for each pair of distinct elements  $a, b \in \Sigma$ , if there exists  $1 \leq j \leq q$  such that  $\Sigma'_j$  contains both  $a$  and  $b$ , then there exists an  $\Sigma_i$  containing both  $a$  and  $b$ . Finally, quotient this preorder by taking distributions to be equal up to permutation.

► **Proposition 16.** There is a Galois insertion  $\text{Ind}_\Sigma \leftrightarrow \text{Dist}_\Sigma$ .

**Proof.** We construct an injective monotone function  $i : \text{Ind}_\Sigma \rightarrow \text{Dist}_\Sigma$ . Let an independence relation  $I$  over  $\Sigma$  be given, with induced dependence relation  $D_I$ . Construct the undirected *dependency graph*: vertices are elements of  $\Sigma$  and there is an edge  $(a, b)$  for every  $(a, b) \in D_I$ . Choose an ordering of maximal cliques of  $D_I$ , and define a distributed alphabet by taking  $\Sigma_i$  to be the elements of  $\Sigma$  in the maximal clique  $i$ . Different orderings give the same distribution up to permutation, and so the same element of  $\text{Dist}_\Sigma$ . This is injective since distinct independence relations induce distinct dependency graphs. It is monotone since if  $I \subseteq I'$  then the dependency graph  $D_I$  is at least as connected as  $D_{I'}$ , so if  $a, b$  both belong to a maximal clique of  $D_{I'}$  then they will both belong to a maximal clique of  $D_I$ .

We construct a monotone function  $r : \text{Dist}_\Sigma \rightarrow \text{Ind}_\Sigma$ . Let  $(\Sigma_1, \dots, \Sigma_k)$  be a distribution. Define a relation  $I$  by  $(a, b) \in I \iff \text{loc}(a) \cap \text{loc}(b) = \emptyset$ . This is irreflexive and symmetric, and so an independence relation.  $r$  is also clearly well-defined and monotone. Finally it is easy to check that  $r \circ i : \text{Ind}_\Sigma \rightarrow \text{Ind}_\Sigma$  is the identity. ◀

Put otherwise, though the same independence relation may be induced by many different distributions, independence relations correspond bijectively with the distributions in the image of  $i \circ r$ , that is, the distributions obtained via the maximal clique construction.

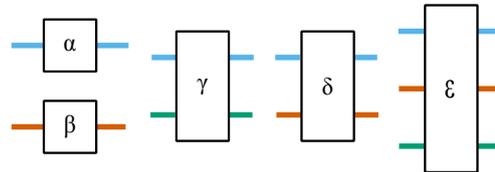
### 4.2 Symmetric monoidal languages over monoidal distributed alphabets

We now turn to the interpretation of these notions in terms of symmetric monoidal languages. A distribution can be seen as a monoidal graph in which sorts are the locations (runtimes).

► **Definition 17.** *A monoidal distributed alphabet is a finite monoidal graph  $\Gamma$  with the following properties:*

- $\Gamma$  has set of sorts a finite ordinal  $S_\Gamma = \{1 < 2 < \dots < k\}$  for  $k \geq 1$ ,
- sorts  $i \in S_\Gamma$  appear in order in the sources and targets of each generator  $\gamma \in B_\Gamma$ ,
- each sort  $i \in S_\Gamma$  appears at most once in each source and target,
- for each generator  $\gamma \in B_\Gamma$ , the sources and targets are non-empty and equal:  $s(\gamma) = t(\gamma)$ .

In brief, every generator in the alphabet is equipped with some set of runtimes, which serve as its source and target, and the runtimes are conserved. Figure 3 gives an example.



■ **Figure 3** An example of a monoidal distributed alphabet. For example,  $\delta$  and  $\beta$  are independent but  $\gamma$  and  $\alpha$  are not. We use colours for clarity, here blue = 1 < red = 2 < green = 3.

This gives us a way of representing distributions as monoidal graphs and vice-versa, if the graph is a monoidal distributed alphabet. Following Proposition 15, we will use  $\text{loc}(\boxed{\gamma})$  to mean the arity (= coarity) of a generator  $\boxed{\gamma}$ . Since we choose a finite ordinal for the sorts, we have that:

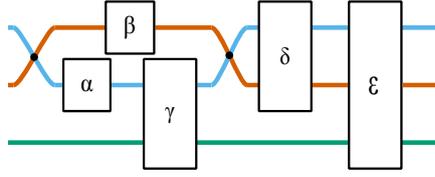
► **Proposition 18.** *Distributed alphabets are in bijection with monoidal distributed alphabets.*

Since the *ordering* of the runtimes is ultimately not relevant to the structure of a trace, we should allow them to freely cross each other in our string diagrams: this is precisely what is enabled by taking the *symmetric* monoidal languages over these alphabets. We also need each runtime to appear once in each element of these languages, so we take the boundaries to be  $1 \otimes \dots \otimes n$ , which we will write as  $\frac{1}{n}$ .

► **Definition 19.** *A monoidal trace language is a symmetric monoidal language of the form  $L \subseteq \mathcal{F}_\times \Gamma \left( \frac{1}{n}, \frac{1}{n} \right)$  where  $\Gamma$  is a monoidal distributed alphabet.*

Figure 4 gives an example of an element in a monoidal trace language over the monoidal distributed alphabet in Figure 3. We call such morphisms *monoidal traces*, and indeed we shall see below that they are exactly Mazurkiewicz traces. The corresponding string diagram gives an intuitive representation of traces as topological objects.

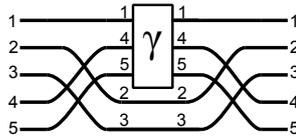
We now show that monoidal trace languages correspond precisely to Mazurkiewicz trace languages (Theorem 22), by establishing an isomorphism of monoids between trace monoids and monoids of string diagrams generated by monoidal distributed alphabets. Fix a monoidal distributed alphabet  $\Gamma$ . Recall that endomorphism hom-sets in a category are monoids under composition, and that the hom-set  $\mathcal{F}_\times \Gamma \left( \frac{1}{n}, \frac{1}{n} \right)$  has elements string diagrams  $\frac{1}{n} \rightarrow \frac{1}{n}$  over  $\Gamma$ .



■ **Figure 4** An example of a monoidal trace.  $\beta$  is independent of  $\alpha$  and  $\gamma$ , but not  $\delta$  or  $\epsilon$ . Thus  $\alpha\gamma\beta\delta\epsilon$  and  $\beta\alpha\gamma\delta\epsilon$  are two possible serializations of this trace, corresponding to sliding  $\beta$  past  $\alpha$  and  $\gamma$  in the string diagram. We use colours for sorts, blue = 1 < red = 2 < green = 3.

- **Lemma 20.** *The hom-set  $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$  admits the following presentation as a monoid:*
- *Generators:* For each  $\boxed{\gamma} \in \Gamma$ , the string diagram  $N(\gamma) : 1 \otimes \dots \otimes n \rightarrow 1 \otimes \dots \otimes n$  built from symmetries, followed by  $\boxed{\gamma}$  tensored with identities, followed by the inverse symmetry. See Figure 5 for an example.
  - *Equations:*  $N(\alpha) \circledast N(\beta) = N(\beta) \circledast N(\alpha) \iff \text{loc}(\boxed{\alpha}) \cap \text{loc}(\boxed{\beta}) = \emptyset$ , where  $\circledast$  denotes composition of string diagrams in diagrammatic (left-to-right) order.

**Proof.** We construct an isomorphism between the monoids. Let  $s \in \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$  be a string diagram. We can use interchange (Figure 1) to impose a linear order of generators from left to right in the diagram, e.g.  $\boxed{\gamma_1}, \dots, \boxed{\gamma_n}$ . This is called putting  $s$  in *general position*, by perturbing generators at the same horizontal position [16]. We then split the string diagram into a sequence of slices, each containing one generator. For a slice with right (or left) boundary  $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix}$ , we can use the permutation  $\begin{smallmatrix} k_1 \\ \vdots \\ k_n \end{smallmatrix} \rightarrow \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}$  followed by its inverse (or vice-versa) to finally obtain  $s$  as a sequence  $N(\gamma_1) \circledast \dots \circledast N(\gamma_n)$ . Any other possible sequence of generators is obtainable by repeatedly interchanging generators: this is possible if and only if their locations are disjoint. Consequently, this defines a function from  $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$  to the monoid presented above. Given that, as argued above, the slicing construction is unique up to interchanging independent generators, this defines a homomorphism. Conversely, given a generator  $N(\gamma)$  in the presentation, we map this to the same string diagram in  $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ . Again, it follows from interchange that this extends to a homomorphism, inverse to that above. ◀



■ **Figure 5** An example of a generator  $N(\gamma)$  as in Lemma 20.

We now show that trace monoids are isomorphic to the endomorphism monoids  $\mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ .

- **Lemma 21.** *Let  $I$  be an independence relation on an alphabet  $\Sigma$ , and  $\Gamma$  the monoidal distributed alphabet induced by the corresponding distribution (Proposition 18). Then there is an isomorphism of monoids  $\mathcal{T}(\Sigma, I) \cong \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right)$ .*

**Proof.** We use the presentation of the endomorphism monoid given in Lemma 20. Define a homomorphism  $\alpha : \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}\right) \rightarrow \mathcal{T}(\Sigma, I)$  by mapping generators  $N(\gamma) \mapsto [\gamma]$ . Let  $N(\gamma) \circledast N(\gamma') = N(\gamma') \circledast N(\gamma)$ , then it follows  $[\gamma\gamma'] = [\gamma'\gamma]$  in  $\mathcal{T}(\Sigma, I)$ , since the former holds

iff  $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$ , and so this extends to a homomorphism. Define a homomorphism  $\beta : \mathcal{T}(\Sigma, I) \rightarrow \mathcal{F}_\times \Gamma \left( \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix} \right)$  by mapping generators  $[\gamma] \mapsto N(\gamma)$ .  $[\gamma\gamma'] = [\gamma'\gamma]$  holds iff  $\text{loc}(\gamma) \cap \text{loc}(\gamma') = \emptyset$ , iff  $\text{loc}(\boxed{\gamma}) \cap \text{loc}(\boxed{\gamma'}) = \emptyset$ , iff  $N(\gamma) \circ N(\gamma') = N(\gamma') \circ N(\gamma)$ . Finally it is clear that  $\alpha$  and  $\beta$  are inverses, and so witness an isomorphism of monoids. ◀

The following theorem is now immediate: given a monoidal trace language  $L \subseteq \mathcal{F}_\times \Gamma \left( \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix}, \begin{smallmatrix} 1 \\ \vdots \\ n \end{smallmatrix} \right)$  we obtain a trace language  $L' \subseteq \mathcal{T}(\Sigma, I)$  using the isomorphism, and vice-versa:

► **Theorem 22.** *Monoidal trace languages are exactly Mazurkiewicz trace languages.*

Lemma 21 also shows that composition of traces corresponds simply to concatenation of the corresponding monoidal traces. Diagrams like Figure 4 are commonplace in the trace literature [11, 32]. Theorem 22 gives a formal basis for these diagrams as elements of symmetric monoidal languages.

► **Remark 23.** The idea of monoidal categories with a runtime is made precise by string diagrams for the *effectful categories* of Román [28]. Free props over monoidal distributed alphabets, considered as monoidal categories with *multiple runtimes* suggest a further generalization of effectful categories, sketched as a setting for concurrency by Jeffrey [14, Section 9.4]. We return to this in Section 7, where effectful (premonoidal) categories will be used to equip a trace language with a new runtime that enforces a strict ordering of events.

## 5 Symmetric Monoidal Automata

*Monoidal automata* give an alternative specification of the class of regular monoidal languages: they are analogues of finite-state automata in which transitions have multiple inputs and multiple outputs. Our paper [12] introduced monoidal automata for single-sorted, planar monoidal languages. However, the same data specifies an acceptor for single-sorted symmetric monoidal languages, if we inductively extend to *props*, rather than planar monoidal categories.

In this section we introduce monoidal automata over *multi-sorted* monoidal graphs and show how these recognize (multi-sorted) symmetric monoidal languages. In Section 6, we will see that the asynchronous automata of Zielonka [32] are a natural class of symmetric monoidal automata: those over monoidal distributed alphabets.

► **Definition 24.** *A non-deterministic monoidal semi-automaton is:*

- an input alphabet, given by a finite monoidal graph  $\Gamma$ ,
- an family of non-empty, finite state sets  $\{Q_c\}_{c \in S_\Gamma}$  indexed by the sorts of  $\Gamma$ ,
- for each  $\gamma : c_1 \dots c_n \rightarrow c'_1 \dots c'_m$  in  $\Gamma$ , a transition function  $\Delta_\gamma : \prod_{i=1}^n Q_{c_i} \rightarrow \mathcal{P}(\prod_{j=1}^m Q_{c'_j})$ .

As noted in Section 3, there are several candidates for a notion of initial/final state. In the following, we take initial and final words  $i, f$  over  $\prod Q_c$ . A monoidal semi-automaton equipped with initial and final words turns it into a (non-deterministic) *monoidal automaton*.

For classical NFAs, the assignment  $a \mapsto \Delta_a$  extends uniquely to a functor  $\Sigma^* \rightarrow \text{Rel}$ , the inductive extension of the transition structure from letters to words. We can similarly extend monoidal automata to string diagrams. First, we define the codomain prop,  $\text{Rel}_{\Gamma, Q}$ :

► **Definition 25.** *For a family of sets  $\{Q_c\}_{c \in S_\Gamma}$  indexed by the sorts of  $\Gamma$  then  $\text{Rel}_{\Gamma, Q}$  is the prop with:*

- set of objects  $S_\Gamma^*$ ,
- morphisms  $c_1 \dots c_n \rightarrow c'_1 \dots c'_m$  functions  $\prod_{i=1}^n Q_{c_i} \rightarrow \mathcal{P}(\prod_{j=1}^m Q_{c_j})$ ,

- composition is the usual composition of relations, i.e.  $f \circ g := \mu \circ \mathcal{P}(g) \circ f$ , where  $\mu$  is the canonical map from sets of subsets to subsets,
- $\otimes$  is given on objects by concatenation,
- and on morphisms  $f : \otimes_i c_i \rightarrow \otimes_j c'_j$  and  $g : \otimes_k d_k \rightarrow \otimes_l d'_l$  by  $f \otimes g := \nabla \circ (f \times g)$ , where  $\nabla$  sends pairs of subsets to their cartesian product,
- symmetries  $\sigma : c_1 c_2 \rightarrow c_2 c_1$  are functions  $Q_{c_1} \times Q_{c_2} \rightarrow \mathcal{P}(Q_{c_2} \times Q_{c_1}) : (q, q') \mapsto \{(q', q)\}$ .

Note that a non-deterministic monoidal semi-automaton amounts to a morphism of monoidal graphs  $\Gamma \rightarrow \mathcal{U} \text{Rel}_{\Gamma, Q}$ . The adjunction  $\mathcal{F}_X \dashv \mathcal{U}$  implies that there is a unique extension to a strict monoidal functor  $\mathcal{F}_X \Gamma \rightarrow \text{Rel}_{\Gamma, Q}$ , which we call a non-deterministic symmetric monoidal semi-automaton. This functor maps a string diagram to a relation. When this relation relates the initial word to the final word, the string diagram is *accepted*:

► **Definition 26.** Let  $\Delta : \mathcal{F}_X \Gamma \rightarrow \text{Rel}_{\Gamma, Q}$  be a non-deterministic monoidal automaton with initial and final states  $i, f \in S_\Gamma^*$ . Then the symmetric monoidal language accepted by  $\Delta$  is the set of morphisms  $\mathcal{L}(\Delta) := \{\alpha \in \mathcal{F}_X \Gamma \mid f \in \Delta(\alpha)(i)\}$ .

Intuitively, a run of a symmetric monoidal automaton starts with a *word* of states, whose subwords are modified by transitions corresponding to generators. Identity wires do not modify the states, and symmetries permute adjacent states.

► **Observation 27.** There is an evident correspondence between non-deterministic monoidal automata and regular monoidal grammars. The graphical representation of a grammar (such as Figure 2) makes this most clear: it can also be thought of as the “transition graph” of a non-deterministic monoidal automaton.

► **Remark 28.** We can further abstract our definition of monoidal automaton by noting that  $\text{Rel}_{\Gamma, Q}$  is a sub-prop of the Kleisli category of the powerset monad  $\mathcal{P}$ , and that this monad could be replaced by another commutative monad [27, Corollary 4.3]. For example, replacing  $\mathcal{P}$  with the maybe monad, we obtain deterministic monoidal automata.

## 6 Asynchronous Automata as Symmetric Monoidal Automata

Asynchronous automata were introduced by Zielonka [32] as a true-concurrent operational model of recognizable trace languages, a well-behaved subclass of trace languages analogous to regular languages. In this section we show they are precisely symmetric monoidal automata over monoidal distributed alphabets, which leads to the following theorem:

► **Theorem 29.** Recognizable trace languages are exactly regular symmetric monoidal languages over monoidal distributed alphabets.

We recall the definition of asynchronous automata, before turning to monoidal automata.

► **Definition 30 (Asynchronous automaton [32]).** Let  $(\Sigma_1, \dots, \Sigma_k)$  be a distribution of an alphabet  $\Sigma$ . For each  $1 \leq i \leq k$ , let  $Q_i$  be a non-empty finite set of states, and for each  $\sigma \in \Sigma$  take a transition relation  $\Delta_\sigma : \prod_{i \in \text{loc}(\sigma)} Q_i \rightarrow \mathcal{P}(\prod_{i \in \text{loc}(\sigma)} Q_i)$ . This defines a global transition relation on the set  $Q := \prod_{i=1}^k Q_i$  as follows:

$(q_1, \dots, q_k) \xrightarrow{\sigma} (q'_1, \dots, q'_k) \iff q_i = q'_i \text{ for } i \notin \text{loc}(\sigma) \text{ and } (q'_{i_1}, \dots, q'_{i_j}) \in \Delta_\sigma(q_{i_1}, \dots, q_{i_j})$   
where  $\{i_1, \dots, i_j\} \in \text{loc}(\sigma)$ . Finally let  $\vec{i} \in Q, F \subseteq Q$  be initial and final words of states.

The global transition relation for  $\sigma$  leaves unchanged those states at locations in the complement of  $\text{loc}(\sigma)$ , and otherwise acts according to the local transition  $\Delta_\sigma$ . An asynchronous automaton has a language over  $\Sigma$  given by the extension of the transition relation

to words. Moreover, asynchronous automata have a language of Mazurkiewicz traces over the distribution of  $\Sigma$ : a trace in  $\mathcal{T}(\Sigma, I)$  is accepted when all of its serializations are accepted, which happens when one of its serializations is accepted [32, p. 109]. *Recognizable trace languages* are defined algebraically as those whose syntactic congruence is of finite index [32]. Zielonka’s theorem says that they also have an operational characterization:

► **Theorem 31** (Zielonka [32]). *Asynchronous automata accept precisely the recognizable trace languages.*

Definition 30 closely resembles that of symmetric monoidal automata. Indeed, asynchronous automata are precisely symmetric monoidal automata over monoidal distributed alphabets:

► **Proposition 32.** *For an asynchronous automaton  $\mathcal{A}$ , there is a symmetric monoidal automaton over a monoidal distributed alphabet with the same trace language, and vice-versa.*

**Proof.** An asynchronous automaton with multiple final state words can be normalized to a single final state word in the usual way by introducing a new final state word and modifying transitions appropriately. Then a symmetric monoidal automaton can be constructed by taking the monoidal distributed alphabet associated to the distribution of  $\Sigma$  (Proposition 18), the same transition relations, initial and final state words. We show that the languages coincide. Let  $w \in \mathcal{L}(\mathcal{A})$ , and consider the corresponding trace  $[w]$ . Using Lemma 21, we can produce the corresponding monoidal trace. By construction, this is accepted by the symmetric monoidal automaton defined above. The converse is analogous. ◀

As a corollary, we can invoke Theorem 31 to obtain Theorem 29. In contrast to asynchronous automata, the constructed symmetric monoidal automaton directly accepts traces qua string diagrams, rather than a language of words corresponding to a trace language.

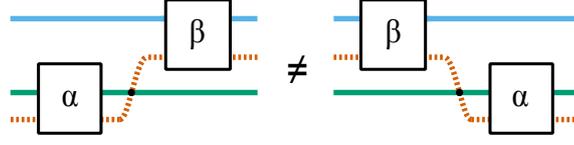
► **Observation 33.** *Jesi, Pighizzini, and Sabadini [15] introduced probabilistic asynchronous automata. Initial and final states, and transition relations are replaced by initial and final distributions, and stochastic transitions. These are precisely what are obtained if the powerset monad in our definition of non-deterministic monoidal automaton (Remark 28) is replaced with the distribution monad [26], whose Kleisli category has morphisms stochastic matrices.*

## 7 Serialization via Premonoidal Categories

Trace theorists often consider trace languages to be word languages with the property of *trace-closure* with respect to an independence relation [19]: if  $u \in L$  and  $u \equiv_I v$  then  $v \in L$ . These languages arise as preimages of trace languages along the quotient map  $q_{\Sigma, I} : \Sigma^* \rightarrow \mathcal{T}(\Sigma, I)$ . For  $L \subseteq \mathcal{T}(\Sigma, I)$  a trace language,  $q_{\Sigma, I}^{-1}(L) \subseteq \Sigma^*$  is its *flattening* or *serialization*.

In this section we show that the serialization of monoidal trace languages can be carried out using the algebra and string diagrams of symmetric premonoidal categories. Premonoidal categories are like monoidal categories, except interchange (Figure 1) does not hold in general. The free (symmetric) premonoidal category on a monoidal graph was described using string diagrams by Román [28]. The idea is simple: the string diagrams are the same as for props, but an extra string (the “runtime”) threads through each generator, preventing interchange. Figure 6 shows two premonoidal morphisms  $\bullet \otimes \bullet \rightarrow \bullet \otimes \bullet$  that are not equal:

In Appendix A, we explain in more detail the construction of the free symmetric premonoidal category  $\mathcal{F}_p\Gamma$  on a monoidal graph  $\Gamma$  using string diagrams. In particular, the runtime string appears only once in each string diagram, reflecting that premonoidal categories do not have a tensor product on morphisms. The endomorphism monoid  $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$  is now the free monoid over the boxes of  $\Gamma$ , since the runtime prevents interchange:



■ **Figure 6** In the free premonoidal category over a monoidal graph, generators are augmented by a string on a new object called the *runtime* (dashed red). This prevents interchange (cf. Figure 1).

► **Proposition 34.** *Let  $\Gamma$  be a monoidal distributed alphabet. Then  $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \cong B_\Gamma^*$ , where  $B_\Gamma$  is the set of boxes of  $\Gamma$ .*

**Proof (Sketch).** By augmenting the generators of  $\Gamma$  with a new runtime, we create a monoidal distributed alphabet in which every generator depends on every other, that is, the independence relation is empty. Thus the corresponding trace monoid is simply  $B_\Gamma^*$ . From here, we can follow the idea of Lemma 21. ◀

We can define a morphism of monoids  $q_\Gamma : \mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \rightarrow \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$  by presenting  $\mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right)$  as in Lemma 20, and defining  $q_\Gamma$  on generators by erasing the runtime string. Theorem 35 then follows immediately from the definitions along with Lemma 21 and Proposition 34:

► **Theorem 35.** *For every alphabet  $B_\Gamma$ , the following square of monoid homomorphisms commutes, where  $q$  is the quotient monoid homomorphism.*

$$\begin{array}{ccc} B_\Gamma^* & \xrightarrow{q} & \tau(B_\Gamma, I) \\ \cong \downarrow & & \downarrow \cong \\ \mathcal{F}_p\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) & \xrightarrow{q_\Gamma} & \mathcal{F}_\times\Gamma\left(\begin{smallmatrix} 1 & 1 \\ \vdots & \vdots \\ n & n \end{smallmatrix}\right) \end{array}$$

As a result, the preimage of a monoidal trace language under the morphism  $q_\Gamma$  corresponds to the serialization of that language.

## 8 Conclusion

There are several directions in which our theory could be developed. A semi-independence relation drops symmetry from an independence relation: it is simply an irreflexive relation. This gives rise to the theory of semicommutations [7], in which *directed* commutations may occur e.g.  $ab \rightarrow ba$ , but not vice-versa. This allows for a more fine-grained specification of concurrency. In terms of monoidal languages, it suggests consideration of monoidal distributed alphabets in which the sources and targets of generators may differ.

As noted in Remark 23, our treatment of trace languages suggests a generalization of the notion of *effectful category* [28] (which include premonoidal categories), in which there are *multiple runtimes*. This would enable a semantics for concurrent systems in which we can consider not only *atomic* actions, but also actions with input and output types. We plan to pursue this axiomatically in future work.

Mazurkiewicz originally introduced traces to give semantics to Petri nets, and showed that this semantics is compositional with respect to *synchronization* of traces [21]. Petri nets have been given semantics in monoidal categories [2, 22], and so the precise relationship of our monoidal formulation of traces to Petri nets remains to be worked out. In particular, this would involve understanding trace synchronization in terms of monoidal categories.

Finally, proofs of Zielonka’s theorem (Theorem 31, see [32] for details) remain highly technical, despite several simplifications since Zielonka’s version. Investigation of whether the algebra of monoidal categories might yield further simplifications is an intriguing direction.

---

## References

- 1 John C. Baez, Brandon Coya, and Franciscus Rebro. Props in network theory. *Theory and Applications of Categories*, 33(25):727–783, 2018.
- 2 John C Baez, Fabrizio Genovese, Jade Master, and Michael Shulman. Categories of nets. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.
- 3 Guillaume Boisseau and Pawel Sobocinski. String diagrammatic electrical circuit theory. *Electronic Proceedings in Theoretical Computer Science*, 372:178–191, 2022.
- 4 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’15, pages 515–526, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2676726.2676993.
- 5 Francis Bossut, Max Dauchet, and Bruno Warin. A Kleene theorem for a class of planar acyclic graphs. *Inf. Comput.*, 117:251–265, March 1995. doi:10.1006/inco.1995.1043.
- 6 Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, 1993. doi:10.1016/0304-3975(93)90054-W.
- 7 M Clerbout, M Latteux, and Y Roos. Semi-commutations. In V Diekert and G Rozenberg, editors, *The Book of Traces*. World Scientific, 1995.
- 8 Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Information and Computation*, 250:59–86, 2016. Quantum Physics and Logic. doi:10.1016/j.ic.2016.02.008.
- 9 Bob Coecke and Aleks Kissinger. *Picturing quantum processes : a first course in quantum theory and diagrammatic reasoning*. Cambridge University Press, 2017.
- 10 V Diekert and G Rozenberg. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
- 11 Volker Diekert and Anca Muscholl. On distributed monitoring of asynchronous systems. In Luke Ong and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, pages 70–84, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 12 Matthew Earnshaw and Pawel Sobociński. Regular Monoidal Languages. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.44.
- 13 Hoogeboom H J and G Rozenberg. Dependence graphs. In V Diekert and G Rozenberg, editors, *The Book of Traces*. World Scientific, 1995.
- 14 Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint*, 1998.
- 15 S. Jesi, G. Pighizzini, and N. Sabadini. Probabilistic asynchronous automata. *Mathematical systems theory*, 29(1):5–31, February 1996. doi:10.1007/BF01201811.
- 16 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.
- 17 C. Krattenthaler. The theory of heaps and the Cartier-Foata monoid. In P. Cartier and D. Foata, editors, *Commutation and Rearrangements*. European Mathematical Information Service, 2006.
- 18 Elena Di Lavore, Giovanni de Felice, and Mario Román. Coinductive streams in monoidal categories, 2022. arXiv:2212.14494.
- 19 Hendrik Maarand and Tarmo Uustalu. Reordering derivatives of trace closures of regular languages. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

- 20 Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965.
- 21 Antoni Mazurkiewicz. Basic notions of trace theory. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 285–363, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.
- 22 José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, 1990. doi:10.1016/0890-5401(90)90013-8.
- 23 Madhavan Mukund. Automata on distributed alphabets. In *Modern Applications of Automata Theory*, pages 257–288. World Scientific, 2012. doi:10.1142/9789814271059\_0009.
- 24 Chad Nester. Concurrent Process Histories and Resource Transducers. *Logical Methods in Computer Science*, Volume 19, Issue 1, January 2023. doi:10.46298/lmcs-19(1:7)2023.
- 25 Dusko Pavlovic. Monoidal computer I: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013. Special Issue: Information Security as a Resource. doi:10.1016/j.ic.2013.03.007.
- 26 Paolo Perrone. Distribution monad (nlab entry), 2019. , Last accessed 2023-03-13. URL: <https://ncatlab.org/nlab/show/distribution+monad>.
- 27 John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5), 1997. doi:10.1017/S0960129597002375.
- 28 Mario Román. Promonads and string diagrams for effectful categories. In *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18 - 22 July, 2022*, 2022. doi:10.48550/arXiv.2205.07664.
- 29 P. Selinger. A survey of graphical languages for monoidal categories. In B. Coecke, editor, *New Structures for Physics*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9\_4.
- 30 Gérard Xavier Viennot. Heaps of pieces, I : Basic definitions and combinatorial lemmas. In Gilbert Labelle and Pierre Leroux, editors, *Combinatoire énumérative*, pages 321–350, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.
- 31 R.F.C. Walters. A note on context-free languages. *Journal of Pure and Applied Algebra*, 62(2):199–203, 1989. doi:10.1016/0022-4049(89)90151-5.
- 32 Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 21(2):99–135, 1987.

## A Symmetric Strict Premonoidal Categories and Functors

We recall the definitions of (symmetric) strict premonoidal categories and their functors. For more details, see the papers [27, 28].

► **Definition 36.** *A strict premonoidal category is a category  $\mathcal{C}$  equipped with:*

- *for each pair of objects  $A, B \in \mathcal{C}$  an object  $A \otimes B$ ,*
- *for each object  $A \in \mathcal{C}$  a functor  $A \triangleleft -$  whose action on objects sends  $B$  to  $A \otimes B$ ,*
- *for each object  $A \in \mathcal{C}$  a functor  $- \triangleright A$  whose action on objects sends  $B$  to  $B \otimes A$ , and*
- *a unit object  $I$ ,*

*such that,*

- *for each  $A \in \mathcal{C}$ , strict unitality  $I \otimes A = A = A \otimes I$  holds, and*
- *for each triple  $A, B, C \in \mathcal{C}$ , strict associativity  $A \otimes (B \otimes C) = (A \otimes B) \otimes C$  holds.*

The families of functors  $A \triangleleft -$ ,  $- \triangleright A$  are called the *whiskerings* with  $A$ : in a premonoidal category we do not have a tensor product of morphisms in general, but we can put an identity on either side of a morphism. A morphism  $f : A \rightarrow B \in \mathcal{C}$  is *central* if for every morphism  $g : C \rightarrow D$ ,  $(B \triangleleft g) \circ (f \triangleright C) = (f \triangleright C) \circ (A \triangleleft g)$ , in other words,  $f$  is central if it interchanges with every other morphism  $g$ .

► **Definition 37.** A strict premonoidal category is symmetric if it is further equipped with a natural isomorphism whose components  $c_{A,B} : A \otimes B \rightarrow B \otimes A$  are central and such that  $c_{B,A} \circ c_{A,B} = 1_{A \otimes B}$ .

► **Definition 38.** A strict premonoidal functor  $F : \mathcal{C} \rightarrow \mathcal{D}$  is a functor sending central morphisms to central morphisms and such that  $F(I_{\mathcal{C}}) = I_{\mathcal{D}}$ ,  $F(X \otimes Y) = F(X) \otimes F(Y)$ .

### A.1 String Diagrams for Premonoidal Categories

We recall the construction of the free symmetric strict premonoidal category over a monoidal graph. This is a special case of the construction of free *effectful categories* in [28, Section 2.3].

We first define the *runtime monoidal graph* over a monoidal graph, which augments the generators with a new wire:

► **Definition 39.** Let  $\mathcal{G}$  be a monoidal graph. Let  $R$  be a sort disjoint from  $S_{\mathcal{G}}$ . The runtime monoidal graph  $\mathcal{G}_R$  has sorts  $S_{\mathcal{G}} + \{R\}$  and for each generator  $\gamma : S_1 \dots S_n \rightarrow S'_1 \dots S'_m$  in  $\mathcal{G}$  a generator  $\gamma : RS_1 \dots S_n \rightarrow RS'_1 \dots S'_m$ .

Graphically we can depict  $\mathcal{G}_R$  as in Figure 7 (right):



■ **Figure 7** Left: A monoidal graph  $\mathcal{G}$ . Right: the associated runtime monoidal graph  $\mathcal{G}_R$ , where the new sort  $R$  is drawn as a dashed string.

► **Definition 40.** The symmetric runtime monoidal category is the free prop  $\mathcal{F}_\times \mathcal{G}_R$  on  $\mathcal{G}_R$ .

► **Theorem 41.** The free symmetric strict premonoidal category  $\mathcal{F}_p \mathcal{G}$  on a monoidal graph  $\mathcal{G}$  has set of objects  $S_{\mathcal{G}}$  and a morphism  $S_1 \otimes \dots \otimes S_n \rightarrow S'_1 \otimes \dots \otimes S'_m$  is a morphism  $R \otimes S_1 \otimes \dots \otimes S_n \rightarrow R \otimes S'_1 \otimes \dots \otimes S'_m$  in the symmetric runtime monoidal category.

**Proof.** The proof follows [28, Theorem 2.14], in the case where  $\mathcal{V}$  is empty, and taking instead the free symmetric strict monoidal category. ◀

In particular note that we no longer have a tensor product of morphisms in  $\mathcal{F}_p \mathcal{G}$ , since the runtime must appear only once in each domain and codomain, but we do have whiskerings for each object.

Consequently the string diagrams for morphisms  $A \rightarrow B$  in  $\mathcal{F}_p \mathcal{G}$  are just morphisms  $R \otimes A \rightarrow R \otimes B$  in the symmetric runtime monoidal category [28, Corollary 2.15].



# Upward Translation of Optimal and P-Optimal Proof Systems in the Boolean Hierarchy over NP

Fabian Egidy  

Julius-Maximilians-Universität Würzburg, Germany

Christian Glaßer 

Julius-Maximilians-Universität Würzburg, Germany

Martin Herold 

Max-Planck-Institut für Informatik, Saarbrücken, Germany

---

## Abstract

We study the existence of optimal and p-optimal proof systems for classes in the Boolean hierarchy over NP. Our main results concern DP, i.e., the second level of this hierarchy:

- If all sets in DP have p-optimal proof systems, then all sets in coDP have p-optimal proof systems.
- The analogous implication for optimal proof systems fails relative to an oracle.

As a consequence, we clarify such implications for all classes  $\mathcal{C}$  and  $\mathcal{D}$  in the Boolean hierarchy over NP: either we can prove the implication or show that it fails relative to an oracle.

Furthermore, we show that the sets SAT and TAUT have p-optimal proof systems, if and only if all sets in the Boolean hierarchy over NP have p-optimal proof systems which is a new characterization of a conjecture studied by Pudlák.

**2012 ACM Subject Classification** Theory of computation → Proof complexity; Theory of computation → Oracles and decision trees

**Keywords and phrases** Computational Complexity, Boolean Hierarchy, Proof Complexity, Proof Systems, Oracle Construction

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.44

**Related Version** *Full Version:* <https://arxiv.org/abs/2304.14702>

**Funding** *Fabian Egidy:* supported by the German Academic Scholarship Foundation.

*Martin Herold:* Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 399223600.

## 1 Introduction

This paper contributes to the study of proof systems initiated by Cook and Reckhow [11]. A proof system for a set  $L$  is a polynomial-time computable function  $f$  whose range is  $L$ . Cook and Reckhow motivate the study of proof systems with the  $\text{NP} = \text{coNP}$  question: they consider propositional proof systems (pps), i.e., proof systems for the set of propositional tautologies (TAUT). They show that there exists a pps with polynomially bounded proofs if and only if  $\text{NP} = \text{coNP}$ . This approach to the  $\text{NP} = \text{coNP}$  question is called the Cook-Reckhow program [9]. To obtain  $\text{NP} \neq \text{coNP}$  one can either show that optimal pps (i.e., pps with at most polynomially longer proofs than any other pps) do not exist or show that a specific pps is optimal and has a non-polynomial lower bound on the length of proofs. This connection led to the investigation of upper and lower bounds for different pps [20] as well as the existence of optimal and p-optimal<sup>1</sup> proof systems for general sets.

---

<sup>1</sup> A stronger notion of optimal. We write (p-)optimal when the statement holds using optimal as well as p-optimal.



© Fabian Egidy, Christian Glaßer, and Martin Herold; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 44; pp. 44:1–44:15

Leibniz International Proceedings in Informatics



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

The latter question was explicitly posed by Krajíček and Pudlák [21] in the context of finite consistency. They revealed the following connection between both concepts: optimal pps exist if and only if there is a finitely axiomatized theory  $S$  that proves for every finitely axiomatized theory  $T$  the statement “ $T$  has no proof of contradiction of length  $n$ ” by a proof of polynomial length in  $n$ . If optimal pps exist, then a weak version of Hilbert’s program is possible, i.e., proving the “consistency up to some feasible length of proofs” of all mathematical theories [24]. We refer to Krajíček [19] and Pudlák [26] for details on the relationship between proof systems and bounded arithmetic. More recently, Pudlák [25] draws new connections of (p-)optimal proof systems and statements about incompleteness in the finite domain.

Furthermore, proof systems have shown to be tightly connected to promise classes, especially pps to the class of disjoint NP-pairs, called DisjNP. Initiated by Razborov [27], who showed that the existence of p-optimal pps implies the existence of complete sets in DisjNP, many further connections were investigated. More generally, Köbler, Messner and Torán [18] show that the existence of p-optimal proof systems for sets of the polynomial-time hierarchy imply complete sets for promise classes like UP,  $\text{NP} \cap \text{coNP}$ , and BPP. Beyersdorff, Köbler, and Messner [7] and Pudlák [25] connect proof systems to function classes by showing that p-optimal proof systems for SAT imply complete sets for TFNP. Questions regarding non-deterministic function classes can be characterized by questions about proof systems [7]. Beyersdorff [3, 4, 5, 6], Beyersdorff and Sadowski [8] and Glaßer, Selman, and Zhang [13, 14] show further connections between pps and disjoint NP-pairs.

The above connections to important questions of complexity theory, bounded arithmetic, and promise classes motivate the investigation of the question “which sets do have optimal proof systems” posed by Messner [22]. Krajíček and Pudlák [21] were the first to study sufficient conditions for pps by proving that  $\text{NE} = \text{coNE}$  implies the existence of optimal pps and  $\text{E} = \text{NE}$  implies the existence of p-optimal pps. Köbler, Messner, and Torán [18] improve this result to  $\text{NEE} = \text{coNEE}$  for optimal pps and  $\text{EE} = \text{NEE}$  for p-optimal pps. Sadowski [28] shows different characterizations for the existence of optimal pps, e.g., the uniform enumerability of the class of all easy subsets of TAUT. In certain settings one can prove the existence of optimal proof systems for different classes: e.g., by allowing one bit of advice [10], considering randomized proof systems [16, 15], or using a weak notion of simulation [29].

Messner [22] shows that all nonempty<sup>2</sup> sets in P but not all sets in E have p-optimal proof systems. Similarly, all sets in NP but not all sets in coNE have optimal proof systems. Therefore, when going from smaller to larger complexity classes, there has to be a tipping point such that all sets contained in classes below this point have (p-)optimal proof systems, but some set contained in all classes above this point has no (p-)optimal proof systems. Unfortunately, oracle constructions tell us that for many classes between P and E (resp., NP and coNE) the following holds: with relativizable proofs one can neither prove nor refute that p-optimal (resp., optimal) proof systems exist (e.g. coNP [1, 21] and PSPACE [1, 12]). Thus, with the currently available means it is not possible to precisely locate this tipping point, but we can rule out certain regions for its location. For this, we investigate how the existence of (p-)optimal proof system for all sets of the class  $\mathcal{C}$  “translate upwards” to all sets of a class  $\mathcal{D}$  with  $\mathcal{C} \subseteq \mathcal{D}$ . This rules out tipping points between  $\mathcal{C}$  and  $\mathcal{D}$ .

---

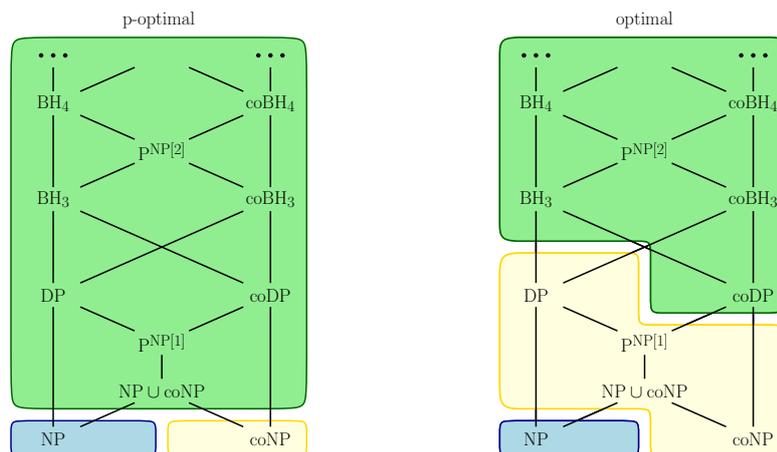
<sup>2</sup> By our definition, FP-functions are total, thus the empty set has no proof system. For the rest of this paper, we omit the word “nonempty” when referring to proof systems for all sets of a class, since this is only a technicality.

**Our Contribution.** Motivated by Messner’s general question, we study the existence of (p-)optimal proof systems for classes inside the Boolean hierarchy over NP. We use the expression “the class  $\mathcal{C}$  has (p-)optimal proof systems” for “all sets of a class  $\mathcal{C}$  have (p-)optimal proof systems”. We say that two classes  $\mathcal{C}$  and  $\mathcal{D}$  are equivalent with respect to (p-)optimal proof systems if  $\mathcal{C}$  has (p-)optimal proof systems if and only if  $\mathcal{D}$  has (p-)optimal proof systems.

For the classes of the Boolean hierarchy over NP, denoted by BH, we identify three equivalence classes for p-optimal proof systems and three other classes for optimal proof systems. We also show that the classes of the bounded query hierarchy over NP are all equivalent for p-optimal proof systems and we identify two equivalence classes for optimal proof systems. Moreover, we show that relativizable techniques cannot prove all identified equivalence classes to coincide. These results follow from our main theorems:

- (i) If DP has p-optimal proof systems, then coDP has p-optimal proof systems.
- (ii) There exists an oracle relative to which coNP has p-optimal proof systems and coDP does not have optimal proof systems.

Using the result by Köbler, Messner, and Torán that (p-)optimality is closed under intersection [18] and two oracles by Khaniki [17], we obtain the equivalence classes visualized in Figure 1, which cannot be proved to coincide with relativizable proofs.



■ **Figure 1** Equivalence classes for p-optimal proof systems (left) and optimal proof systems (right) in the Boolean hierarchy over NP and the bounded query hierarchy over NP.

This clarifies all questions regarding relativizably provable translations of (p-)optimal proof systems for classes in the Boolean hierarchy over NP and the bounded query hierarchy over NP. We cannot expect to prove any further translations with the currently available means, because for every such translation there is an oracle against it. So we are dealing with an interesting situation: while p-optimal proof systems for DP *relativizably imply* p-optimal proof systems for coDP, this does not hold for optimal proof systems. Similarly, all classes of the bounded query hierarchy over NP are equivalent with respect to p-optimal proof systems, but  $P^{NP[1]}$  and  $P^{NP[2]}$  cannot be shown to be equivalent with respect to optimal proof systems by a relativizable proof. The result drastically limits the potential locations of a tipping point in the BH and the bounded query hierarchy over NP. They can only occur between two classes belonging to two different equivalence classes.

Furthermore, our results provide a new perspective on an hypothesis related to feasible versions of Gödel’s incompleteness theorem: Pudlák [25] studies several conjectures about incompleteness in the finite domain by investigating the (un)provability of sentences of some

specific form in weak theories. These conjectures can also be expressed as the non-existence of complete sets in promise classes or non-existence of (p-)optimal proof systems for sets. Pudlák considers the conjecture  $\text{CON} \vee \text{SAT}$  stating that  $\text{TAUT}$  does not have p-optimal proof systems or  $\text{SAT}$  does not have p-optimal proof systems. Khaniki [17] proves this conjecture to be equivalent to  $\text{RFN}_1$ , which is another conjecture considered by Pudlák. Our results show that both conjectures are equivalent to the statement that  $\text{BH}$  does not have p-optimal proof systems.

## 2 Preliminaries

Let  $\Sigma = \{0, 1\}$  be the default alphabet and  $\Sigma^*$  be the set of finite words over  $\Sigma$ . We call subsets of  $\Sigma^*$  languages and sets of languages classes. We denote the length of a word  $w \in \Sigma^*$  by  $|w|$ . The  $i$ -th letter of a word  $w$  for  $0 \leq i < |w|$  is denoted as  $w(i)$ , i.e.,  $w = w(0)w(1) \cdots w(|w| - 1)$ .

The set of all (positive) natural numbers is denoted as  $\mathbb{N}$  ( $\mathbb{N}^+$ ). We write the empty set as  $\emptyset$ . We identify  $\Sigma^*$  with  $\mathbb{N}$  through the polynomial time computable and invertible bijection  $\Sigma^* \rightarrow \mathbb{N}; w \mapsto \sum_{i < |w|} (1 + w(i))2^i$ . This is a variant of the dyadic representation. Thus, we can treat words from  $\Sigma^*$  as numbers from  $\mathbb{N}$  and vice versa, which allows us to use notations, relations and operations of words for numbers and vice versa (e.g. we can define the length of a number by this). We resolve the ambiguity of  $0^i$  and  $1^i$  by always interpreting them as words from  $\Sigma^*$ . The cardinality of a set  $A$  is denoted as  $|A|_c$ . For  $\circ \in \{<, \leq, =, \geq, >\}$ , a set  $A \subseteq \Sigma^*$  and a number  $n \in \mathbb{N}$  we define  $A^{\circ n} = \{w \in A \mid |w| \circ n\}$ . For a clearer notation we use  $\Sigma^{\circ n}$  as  $\Sigma^{*\circ n}$  and  $\Sigma^n$  for  $\Sigma^{=n}$ . The operators  $\cup$ ,  $\cap$ , and  $\setminus$  denote the union, intersection and set-difference. We denote the complement of a set  $A$  relative to  $\Sigma^*$  as  $\bar{A} = \Sigma^* \setminus A$ .

The image of a function  $f$  is denoted as  $\text{img}(f)$ . Let  $\langle \cdot \rangle: \bigcup_{i \geq 0} \mathbb{N}^i \rightarrow \mathbb{N}$  be an injective polynomial time computable and invertible pairing function such that  $|\langle u_1, \dots, u_n \rangle| = 2(|u_1| + \dots + |u_n| + n)$ . The logarithm to the base 2 is denoted as  $\log$ . Furthermore, we define polynomial functions  $p_i: \mathbb{N} \rightarrow \mathbb{N}$  for  $i \in \mathbb{N}^+$  by  $p_i(x) = x^i + i$ .

We use the default model of a Turing machine in the deterministic as well as in the non-deterministic variation, abbreviated by  $\text{DTM}$  and  $\text{NTM}$  respectively. The language decided by a Turing machine  $M$  is denoted as  $L(M)$ . For a number  $s \in \mathbb{N}$  the language of words that are accepted by a Turing machine  $M$  in  $s$  computation steps is denoted as  $L^s(M)$ . We use Turing transducer to compute functions. For a Turing transducer  $F$  we write  $F(x) = y$  when on input  $x$  the transducer outputs  $y$ . A Turing transducer  $F$  computes a total function and we sometimes refer to the function computed by  $F$  as “the function  $F$ ”. Let  $\{F_i\}_{i \in \mathbb{N}}$  and  $\{G_i\}_{i \in \mathbb{N}}$  be standard enumerations of polynomial time Turing transducers. Let  $\{N_i\}_{i \in \mathbb{N}}$  be a standard enumeration of non-deterministic polynomial time Turing machines with the special property that  $N_0$  is the machine that always rejects and  $N_1$  is the machine that always accepts, that is  $L(N_0) = \emptyset$  and  $L(N_1) = \mathbb{N}$ . The runtime of  $F_i$ ,  $G_i$  and  $N_i$  is bounded by  $p_i$ .

► **Proposition 1.** *There is a Turing machine  $M$  and a Turing transducer  $F$  such that for all  $i, s, x \in \mathbb{N}$  the following properties hold:*

- $\langle i, x, 0^s \rangle \in L(M) \Leftrightarrow x \in L^s(N_i)$
- $F(\langle i, x, 0^s \rangle) = \begin{cases} \langle 1, F_i(x) \rangle & \text{if } F_i(x) \text{ stops within } s \text{ steps} \\ \langle 0, 0 \rangle & \text{else} \end{cases}$
- Both machines run in time  $O(|i|s \log s)$ .

FP, P, and NP denote standard complexity classes [23]. For a class  $\mathcal{C}$  define  $\text{co}\mathcal{C} = \{A \subseteq \Sigma^* \mid \overline{A} \in \mathcal{C}\}$ . We define the Boolean hierarchy over NP inductively. Let  $\mathcal{C}$  and  $\mathcal{D}$  be arbitrary complexity classes. First, we define boolean operators on classes:

$$\begin{aligned}\mathcal{C} \wedge \mathcal{D} &= \{A \cap B \mid A \in \mathcal{C} \wedge B \in \mathcal{D}\} \\ \mathcal{C} \vee \mathcal{D} &= \{A \cup B \mid A \in \mathcal{C} \vee B \in \mathcal{D}\}\end{aligned}$$

Then  $\text{BH}_1 = \text{NP}$ ,  $\text{BH}_{2k} = \text{coNP} \wedge \text{BH}_{2k-1}$ ,  $\text{BH}_{2k+1} = \text{NP} \vee \text{BH}_{2k}$ , and  $\text{BH} = \bigcup_{k \geq 1} \text{BH}_k$  where  $\text{BH}_2$  is called DP and BH is called Boolean hierarchy over NP. We want to emphasize that  $\text{DP} = \text{NP} \wedge \text{coNP}$  and  $\text{coDP} = \text{NP} \vee \text{coNP}$ . Wagner [30] showed that  $\text{BH}_k \subseteq \text{BH}_{k+1}$  and  $\text{BH}_k \subseteq \text{coBH}_{k+1}$ . The classes  $\text{P}^{\text{NP}[k]}$  for  $k \in \mathbb{N}^+$  contain all sets that can be accepted by a polynomial time Turing machine that queries at most  $k$  elements from an NP-set. The resulting hierarchy  $\text{P}^{\text{NP}[1]}, \text{P}^{\text{NP}[2]}, \dots$  is called bounded query hierarchy over NP. Beigel [2] shows that  $\text{BH}_{2^k-1} \cup \text{coBH}_{2^k-1} \subseteq \text{P}^{\text{NP}[k]} \subseteq \text{BH}_{2^k} \cap \text{coBH}_{2^k}$ .

We use the common polynomial time many-one reducibility for sets  $A, B \subseteq \Sigma^*$ , i.e.,  $A \leq_m^p B$  if there exists an  $f \in \text{FP}$  such that  $x \in A \Leftrightarrow f(x) \in B$ . For a class  $\mathcal{C}$  and some problem  $A$ , we say that  $A$  is hard for  $\mathcal{C}$  if for all  $B \in \mathcal{C}$  it holds  $B \leq_m^p A$ . The set  $A$  is called complete for  $\mathcal{C}$  if  $A \in \mathcal{C}$  and  $A$  is hard for  $\mathcal{C}$ . We define the following complete problems for NP and DP.

$$\begin{aligned}\mathcal{C} &= \{\langle 0^i, x, 0^p \rangle \mid i \in \mathbb{N}, x \in \Sigma^* \text{ and } x \in L^p(N_i)\} \\ \mathcal{D} &= \{\langle 0^i, 0^j, x, 0^p \rangle \mid i, j \in \mathbb{N}, x \in \Sigma^* \text{ and } x \in L^p(N_i) \cap \overline{L^p(N_j)}\} \\ \mathcal{D}' &= \mathcal{D} \cup \{w \mid \nexists i, j \in \mathbb{N}, x \in \Sigma^* : \langle 0^i, 0^j, x, 0^p \rangle = w\}\end{aligned}$$

It is easy to see that  $\mathcal{C}$  is NP-complete and  $\mathcal{D}$  and  $\mathcal{D}'$  are DP-complete. Furthermore, their complements are complete for coNP and coDP respectively. The purpose of  $\mathcal{D}'$  is that  $\overline{\mathcal{D}'}$  consists only of words of the form  $\langle 0^i, 0^j, x, 0^p \rangle$ , which simplifies some arguments in section 3. Let  $N_{\mathcal{C}}$  denote the polynomial time machine with  $L(N_{\mathcal{C}}) = \mathcal{C}$ .

We use proof systems for sets defined by Cook and Reckhow [11]. They define a function  $f \in \text{FP}$  to be a proof system for  $\text{img}(f)$ . Furthermore:

- A proof system  $g$  is (p-)simulated by a proof system  $f$ , denoted by  $g \leq f$  (resp.,  $g \leq^p f$ ), if there exists a total function  $\pi$  (resp.,  $\pi \in \text{FP}$ ) and a polynomial  $p$  such that  $|\pi(x)| \leq p(|x|)$  and  $f(\pi(x)) = g(x)$  for all  $x \in \Sigma^*$ . In this context the function  $\pi$  is called simulation function. Note that  $g \leq^p f$  implies  $g \leq f$ .
- A proof system  $f$  is (p-)optimal for  $\text{img}(f)$ , if  $g \leq f$  (resp.,  $g \leq^p f$ ) for all  $g \in \text{FP}$  with  $\text{img}(g) = \text{img}(f)$ .
- A complexity class  $\mathcal{C}$  has (p-)optimal proof systems, if every  $A \in \mathcal{C}$  with  $A \neq \emptyset$  has a (p-)optimal proof system.
- We say that (p-)optimal proof systems translate from a class  $\mathcal{C}$  to  $\mathcal{D}$  if the existence of (p-)optimal proof systems for  $\mathcal{C}$  implies their existence for  $\mathcal{D}$ .

By the following result of Köbler, Messner and Torán [18], we can prove or refute the existence of (p-)optimal proof systems for a class  $\mathcal{C}$  by proving or refuting the existence of such proof systems for a complete set of  $\mathcal{C}$ .

► **Proposition 2** ([18]). *If  $A \subseteq \Sigma^*$  has a (p-)optimal proof system and  $\emptyset \neq B \leq_m^p A$ , then  $B$  has a (p-)optimal proof system.*

► **Corollary 3.** *If  $A \subseteq \Sigma^*$  is a hard set for some class  $\mathcal{C}$  and  $A$  has a (p-)optimal proof system, then  $\mathcal{C}$  has (p-)optimal proof systems.*

Furthermore, it was shown by Köbler, Messner, and Torán [18] that the class of sets having (p-)optimal proof systems is closed under intersection. This result can easily be extended to the operator  $\wedge$  for complexity classes.

► **Proposition 4** ([18]). *If  $A, B \subseteq \Sigma^*$ ,  $A \cap B \neq \emptyset$  and both sets have a (p-)optimal proof system, then  $A \cap B$  has a (p-)optimal proof system.*

► **Corollary 5.** *If two classes  $\mathcal{C}$  and  $\mathcal{D}$  have (p-)optimal proof systems, then  $\mathcal{C} \wedge \mathcal{D}$  has (p-)optimal proof systems.*

Finally, every (p-)optimal proof system can be transformed into a (p-)optimal proof system that runs in linear time by polynomially padding the proofs.

► **Proposition 6.** *If  $f$  is a (p-)optimal proof system for  $A \subseteq \Sigma$ , then there is a (p-)optimal proof system  $g$  for  $A$  that runs in linear time.*

### 3 Translation of P-Optimal Proof Systems from DP to coDP

In this chapter we show that p-optimal proof systems for DP imply p-optimal proof systems for coDP. This proof is based on machine simulation which is a relativizable proof technique. Thus, the following theorem also holds in the presence of an arbitrary oracle  $O$ .

► **Theorem 7.** *If there exists a p-optimal proof system for  $D$ , then there exists a p-optimal proof system for  $\overline{D}$ .*

We start by sketching the key idea used in the proof. Our approach needs some technique to verify that a given instance is in  $\overline{D}$ . There is no known way to decide  $\overline{D}$  in polynomial time, but we can use the p-optimal proof system for  $D$  for this verification. We define a function  $f' : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  such that there is a polynomial-time-computable encoding  $c : \mathbb{N} \rightarrow \mathbb{N}$  with  $f'(a, c(x)) \in D$  if and only if  $F_a(x) \in \overline{D}$  for all  $a \in \mathbb{N}$  and  $x \in \mathbb{N}$ . Furthermore,  $f'(a, x)$  can be computed in time  $|x|^{O(a)}$ . We derive a class of functions  $\{f'_a\}_{a \in \mathbb{N}}$  from  $f'$  by fixing the first input to  $a$ . Note that  $f'_a$  runs in polynomial time for a fixed  $a \in \mathbb{N}$  and that  $f'_a$  is a proof system for  $D$  if and only if  $F_a$  is a proof system for  $\overline{D}$ . Now, we define a machine that uses an additional input to verify  $F_a(x) \in \overline{D}$ . The inputs of the machine are  $a, x, b$  and it accepts if and only if  $f(F_b(x)) = f'_a(x)$  for a p-optimal proof system  $f$  of  $D$ . So, if  $F_a$  is a proof system for  $\overline{D}$ , we know  $f'_a$  is a proof system for  $D$ . Thus, by the fact that  $f$  is p-optimal, there is a  $b \in \mathbb{N}$  such that  $f(F_b(x)) = f'_a(x)$  for all  $x \in \mathbb{N}$ . Thus, when knowing the value  $b$ , the machine can verify  $F_a(x) \in \overline{D}$  for all  $x \in \mathbb{N}$  for a proof system  $F_a(x)$  for  $\overline{D}$  by accepting  $f(F_b(c(x))) = f'_a(c(x))$ . On the other hand if  $F_a(x) \notin \overline{D}$  there is no  $b$  such that  $f(F_b(c(x))) = f'_a(c(x))$  because  $f'_a(c(x)) \notin D = \text{img}(f)$ .

**Proof.** We start by defining an NTM  $A$  that checks for given  $a, y'$  whether  $F_a(y') = \langle i, j, x', 0^p \rangle \in \overline{D}$ . Since a deterministic polynomial-time computation cannot check every possible path of a coNP machine  $N_j$ ,  $A$  gets a path  $y$  of  $N_j$  as an additional input and has the property that it accepts for all possible paths  $y$  if and only if  $F_a(y') \in \overline{D}$ . Arguing over all  $y'$  this means if  $F_a$  is a proof system for  $\overline{D}$ , then for all  $y'$  and all corresponding paths  $y$  the machine  $A$  accepts on input  $a, y', y$ .

Let  $f$  be a p-optimal proof system for  $D$ . Without loss of generality we assume  $f(x)$  can be computed in  $O(|x|)$  time by Proposition 6. We define  $A$  on input  $x$  as follows.

- (i) Check whether  $x = \langle a, y, y' \rangle$  for some  $a \in \mathbb{N}$  and  $y, y' \in \Sigma^*$ , otherwise reject.
- (ii) Check whether  $F_a(y') = \langle i, j, x', 0^p \rangle$  with  $i, j, p \in \mathbb{N}$  and  $x' \in \Sigma^*$  and whether  $y \in \Sigma^p$ , otherwise reject.

- (iii) Accept if  $N_i(x')$  does not accept on path  $y$  within  $p$  steps.
- (iv) Simulate the first  $p$  steps of  $N_j(x')$ .
- (v) Accept if the simulation of  $N_j(x')$  accepted within the first  $p$  steps, otherwise reject.

► **Observation 8.**  $A(x)$  runs in time  $O(|x|^{3a})$  for  $x = \langle a, y, y' \rangle$ .

**Proof.** Checking whether  $x$  has the required format is possible in linear time. By Proposition 1,  $F_a(y')$  can be computed in time

$$\begin{aligned} O(|a| \cdot (|y'|^a + a) \log(|y'|^a + a)) &\subseteq O(|a| \cdot (|y'|^a + a)^2) \subseteq O((|x|^a + a)^3) \\ &\subseteq O(|x|^{3a} + a^3) \subseteq O(|x|^{3a}). \end{aligned}$$

The first  $p$  steps of the path  $y$  of  $N_i(x')$  can be simulated in time

$$O(|i| \cdot p \log p) \subseteq O(|F_a(y')| \cdot |F_a(y')|^2) \subseteq O(|x|^{3a})$$

for all  $a \in \mathbb{N}$ . The computation  $N_j(x')$  can be simulated in  $O(|j| \cdot p \log p) \subseteq O(|x|^{3a})$  time for all  $a \in \mathbb{N}$ . ◀

► **Claim 9.** Let  $F_a(y') = \langle i, j, x', 0^p \rangle$ . Then  $F_a(y') \in \overline{D'} \Leftrightarrow \forall y \in \Sigma^p : \langle a, y, y' \rangle \in L(A)$ .

**Proof.** First we show " $\Rightarrow$ ". We consider two cases:

- Suppose  $x' \in L^p(N_i)$ . By  $\langle i, j, x', 0^p \rangle \in \overline{D'}$ , it holds  $x' \in L^p(N_j)$ . The machine  $A$  on input  $\langle a, y, y' \rangle$  with  $F_a(y') = \langle i, j, x', 0^p \rangle \in \overline{D'}$  and  $y \in \Sigma^p$  rejects only, if the non-deterministic check in step (iv) fails. But this is impossible since  $x' \in L^p(N_j)$ .
- Suppose  $x' \notin L^p(N_i)$ . In this case  $N_i(x')$  does not accept within  $p$  steps for all paths  $y \in \Sigma^p$ . Thus, the machine accepts in step (iii).

Now, we show " $\Leftarrow$ ". Again, we distinguish two cases:

- Suppose  $A(\langle a, y, y' \rangle)$  accepts in step (iii) for all  $y \in \Sigma^p$ . This implies that  $x' \notin L^p(N_i)$ . Thus,  $\langle i, j, x', 0^p \rangle \in \overline{D'}$ .
- If  $A$  accepts but not in step (iii), we conclude it accepts in step (v). Hence, it holds  $x' \in L^p(N_j)$  and  $\langle i, j, x', 0^p \rangle \in \overline{D'}$ . ◀

We want to define functions  $f_a$  in such a way that  $f_a$  is a proof system for  $D$  if  $F_a$  is a proof system for  $\overline{D'}$ . For this, we can exploit the relationship of  $A$  to proof systems of  $\overline{D'}$  shown in Claim 9. Specifically,  $f_a$  trusts that  $A(\langle a, y, y' \rangle)$  accepts for specific  $y$  and  $y'$  (note that the accepting behavior of  $A$  has influence on  $D$ ), which is equivalent to  $F_a$  being a proof system for  $\overline{D'}$ .

Choose  $a_1 \in \mathbb{N}$  such that  $N_{a_1} = A$ . Let  $k_A \in \mathbb{N}^+$  be a constant such that  $A(x)$  runs in time  $k_A|x|^{3a} + k_A$ . Recall that  $N_0$  always rejects. We define a function  $f^* : \mathbb{N} \rightarrow \mathbb{N}$ :

$$f^*(x) = \begin{cases} \langle a_1, 0, \langle a, y, y' \rangle, 0^{k_A|x|^{3a}+k_A} \rangle & \text{if } x = \langle a, y, y' \rangle 0 \wedge F_a(y') = \langle i, j, x', 0^p \rangle \\ & \wedge y \in \Sigma^p \\ f(x') & \text{if } x = x' 1 \\ f(0) & \text{else} \end{cases}$$

► **Observation 10.**  $f^*(x)$  runs in  $O(|x|^{3a})$  time for  $x = \langle a, y, y' \rangle 0$ .

**Proof.** The case distinction for the first case is possible in  $O(|x|^{3a})$  time because computing  $F_a(y')$  is possible in that time. The output of the first case with exception of the unary runtime parameter  $0^{k_A \cdot (|x|^{3a}) + k_A}$  is possible in linear time. The unary runtime parameter can be computed in  $O(|x|^{3a})$  time. The output of the other cases is possible in linear time. ◀

We define a function  $f_a$  that is obtained from  $f^*$  by fixing an index  $a \in \mathbb{N}$  of a polynomial-time function  $F_a$ .

$$f_a(x) = \begin{cases} f^*(\langle a, y, y' \rangle 0) & \text{if } x = \langle a, y, y' \rangle 0 \\ f(x') & \text{if } x = x' 1 \\ f(0) & \text{else} \end{cases}$$

► **Observation 11.** For  $a \in \mathbb{N}$  and  $y, y' \in \Sigma^*$  it holds that  $f_a(\langle a, y, y' \rangle 0) = f^*(\langle a, y, y' \rangle 0)$ .

► **Observation 12.** For a fixed  $a \in \mathbb{N}$  the function  $f_a(x)$  can be computed in polynomial time.

**Proof.** This follows from Observation 10 and the linear runtime of  $f$ . ◀

▷ **Claim 13.** Let  $a \in \mathbb{N}$  and  $y' \in \Sigma^*$  such that  $F_a(y') = \langle i, j, x', 0^p \rangle \notin \overline{D'}$ . Then there is a  $y \in \Sigma^p$ , such that  $f_a(\langle a, y, y' \rangle 0) \notin D$ .

**Proof.** By Claim 9 we conclude the existence of a  $y \in \Sigma^p$  such that  $A(\langle a, y, y' \rangle)$  rejects. This implies  $f_a(\langle a, y, y' \rangle 0) = \langle a_1, 0, \langle a, y, y' \rangle, 0^{k_A|x|^{3a}+k_A} \rangle \notin D$ . ◀

▷ **Claim 14.** Let  $a \in \mathbb{N}$ . Then  $\text{img}(F_a) \subseteq \overline{D'} \Leftrightarrow \text{img}(f_a) = D$ .

**Proof.** First we show " $\Rightarrow$ ".  $D \subseteq \text{img}(f_a)$  holds because  $\text{img}(f) \subseteq \text{img}(f_a)$  and  $f$  is a proof system for  $D$ . Let  $x \in \Sigma^*$ . In the bottom two cases of  $f_a$  and  $f^*$  it is easy to see  $f_a(x) \in D$  and  $f^*(x) \in D$ . So we can assume  $x = \langle a, y, y' \rangle 0$  with  $F_a(y') = \langle i, j, x', 0^p \rangle$  and  $y \in \Sigma^p$ . Since  $\text{img}(F_a) \subseteq \overline{D'}$ , it holds  $\langle i, j, x', 0^p \rangle \in \overline{D'}$ . By Claim 9 we obtain that  $\langle a, y, y' \rangle \in L(A)$  for all  $y \in \Sigma^p$ . Thus,  $f_a(x) = f^*(\langle a, y, y' \rangle 0) = \langle a_1, 0, \langle a, y, y' \rangle, 0^{k_A|x|^{3a}+k_A} \rangle \in D$  since for all  $y \in \Sigma^p$ ,  $N_{a_1}(\langle a, y, y' \rangle)$  accepts,  $N_0(\langle a, y, y' \rangle)$  rejects and  $|\langle a, y, y' \rangle| \leq |x|$ .

" $\Leftarrow$ " follows directly as the contraposition of Claim 13. ◀

We want to define another NTM  $B$  that checks for given  $a, y'$  whether  $F_a(y') \in \overline{D'}$ . To achieve this we use Claims 13 and 14.  $B$  checks for all  $y \in \Sigma^p$  whether  $f(F_b(\langle a, y, y' \rangle 0)) = f_a(\langle a, y, y' \rangle 0)$  on input  $a, b, y'$ . So if  $F_a$  is a proof system for  $\overline{D'}$ , then there is a  $b$  such that  $B(a, b, y')$  rejects for all  $y'$ . Furthermore, if  $F_a(y') \notin \overline{D'}$ , then  $B(a, b, y')$  accepts for all  $b$ .  $B(x)$  operates as follows.

- (i) Check whether  $x = \langle a, b, y' \rangle$  for some  $a, b \in \mathbb{N}$  and  $y' \in \Sigma^*$ , otherwise reject.
- (ii) Check whether  $F_a(y') = \langle i, j, x', 0^p \rangle$  for some  $i, j, p \in \mathbb{N}$  and  $x' \in \Sigma^*$ , otherwise reject.
- (iii) Branch non-deterministically every  $y \in \Sigma^p$ .
- (iv) Accept if  $f(F_b(\langle a, y, y' \rangle 0)) \neq f_a(\langle a, y, y' \rangle 0)$ .
- (v) Reject.

► **Observation 15.**  $B(x)$  runs in time  $O(|x|^{9a^2b})$  for  $x = \langle a, b, y' \rangle$ .

**Proof.** Checking whether the input is formatted correctly is possible in linear time.  $F_a(y')$  can be computed in  $O(|a| \cdot (|y'|^a + a) \log(|y'|^a + a)) \subseteq O(|x|^{3a})$  time. We also observe that  $|F_a(y')| \leq |y'|^a + a$ . In line (iv) it holds that  $|F_a(y')| \geq 2p \geq 2|y|$  and in line 1 it holds that  $2|a| + 2|y'| + 8 \leq |x|$ , and therefore,

$$|\langle a, y, y' \rangle 0| = 2(|a| + |y| + |y'| + 3) + 1 \leq |F_a(y')| + |x| \leq |x|^a + a + |x| \leq |x|^{a+1} + a.$$

Thus, by Observation 10, computing  $f_a(\langle a, y, y' \rangle 0)$  is possible in time

$$O((|x|^{a+1} + a)^{3a}) \subseteq O(|x|^{a^2+3a}) \subseteq O(|x|^{3a^2+6a}) \subseteq O(|x|^{9a^2}).$$

The value  $F_b(\langle a, y, y' \rangle 0)$  can be computed in time

$$\begin{aligned} O(|b| \cdot ((|x|^{a+1} + a)^b + b) \log((|x|^{a+1} + a)^b + b)) &\subseteq O(|b| \cdot (|x|^{a+2})^b + b) \\ &\quad \cdot \log((|x|^{a+2})^b + b) \\ &\subseteq O(|b| \cdot (|x|^{ab+2b} + b) \log(|x|^{ab+2b} + b)) \\ &\subseteq O(|b| \cdot (|x|^{ab+2b+1}) \log(|x|^{ab+2b+1})) \\ &\subseteq O(|b| \cdot |x|^{2ab+4b+1}) \subseteq O(|x|^{2ab+4b+2}). \end{aligned}$$

In particular  $|F_b(\langle a, y, y' \rangle 0)| \in O(|x|^{2ab+4b+2})$  and hence the computation of  $f(F_b(\langle a, y, y' \rangle 0))$  is possible in time  $O(|x|^{2ab+4b+2})$ . We simplify the sum of these runtimes.

$$O(|x|^{3a} + |x|^{9a^2} + |x|^{2ab+4b+2}) \subseteq O(|x|^{9a^2b}) \quad \blacktriangleleft$$

▷ **Claim 16.** Let  $a \in \mathbb{N}$  and  $y' \in \Sigma^*$ , such that  $F_a(y') = \langle i, j, x', 0^p \rangle \notin \overline{D'}$ . Then for all  $b \in \mathbb{N}$  it holds that  $\langle a, b, y' \rangle \in L(B)$ .

*Proof.* By Claim 13 there is a  $y \in \Sigma^p$  such that  $f_a(\langle a, y, y' \rangle 0) \notin D = \text{img}(f)$ . Thus,  $B$  accepts in step (iv), because  $\text{img}(f) = D$ .  $\blacktriangleleft$

▷ **Claim 17.** Let  $a \in \mathbb{N}$ , such that  $\text{img}(F_a) \subseteq \overline{D'}$ . Then, there is some  $b \in \mathbb{N}$ , such that for all  $y' \in \Sigma^*$  it holds that  $\langle a, b, y' \rangle \notin L(B)$ .

*Proof.* By Claim 14 and Observation 12, we know that  $f_a$  is a proof system for  $D$ . Since  $f$  is a p-optimal proof system for  $D$ , there exists some  $b \in \mathbb{N}$ , such that for all  $\hat{x} \in \Sigma^*$  it holds that  $f_a(\hat{x}) = f(F_b(\hat{x}))$ . Thus, the computation  $B(\langle a, b, y' \rangle)$  cannot accept in step (iv) independent of  $y'$ . Hence, the machine rejects.  $\blacktriangleleft$

Now, we define a function  $g_{a,b}$  for every pair of possible proof system  $F_a$  and possible simulation function  $F_b$ . Similarly to  $f_a$ , the function  $g_{a,b}$  trusts that  $B(\langle a, b, y' \rangle)$  accepts for all  $y'$  (note that the accepting behavior of  $B$  has influence on  $D$ ). If  $F_a$  is a proof system for  $\overline{D'}$ , then there is a  $b \in \mathbb{N}$  such that  $g_{a,b}$  is a proof system for  $D$  because the machine  $B$  accepts on input  $a, b, y'$  for all  $y' \in \Sigma^*$ . For  $F_a(y') \notin \overline{D'}$ , we know there is no  $b \in \mathbb{N}$  such that  $B$  accepts on input  $a, b, y'$ . Hence, the corresponding output of  $g_{a,b}$  is not in  $D$ .

Let  $b_1$  be the index of the NTM  $B$ , that is  $N_{b_1} = B$ . Furthermore, let  $k_B \in \mathbb{N}^+$  be a constant such that  $B(x)$  runs in time  $k_B|x|^{9a^2b} + k_B$  for all  $x \in \Sigma^*$ . Recall that  $N_1$  always accepts. We define a function  $g : \mathbb{N} \rightarrow \mathbb{N}$  whose input consists of two indices  $a, b \in \mathbb{N}$  of polynomial-time functions  $F_a, F_b$  and a proof  $y' \in \Sigma^*$ .

$$g(x) = \begin{cases} \langle 1, b_1, \langle a, b, y' \rangle, 0^{k_B|x|^{9a^2b} + k_B} \rangle & \text{if } x = \langle a, b, y' \rangle 0 \\ f(x') & \text{if } x = x' 1 \\ f(0) & \text{else} \end{cases}$$

► **Observation 18.**  $g(x)$  runs in  $O(|x|^{9a^2b})$  time for  $x = \langle a, b, y' \rangle 0$ .

**Proof.** Checking whether the input is formatted correctly is possible in linear time. Furthermore, the output with exception of the last entry of the list can be computed in linear time. The string  $0^{k_B|x|^{9a^2b} + k_B}$  can be computed in  $O(|x|^{9a^2b})$  time.  $\blacktriangleleft$

## 44:10 Upward Translation of (P-)Optimal Proof Systems in the Boolean Hierarchy over NP

We define a function  $g_{a,b} : \mathbb{N} \rightarrow \mathbb{N}$  that is obtained from  $g$  by fixing two indices  $a, b \in \mathbb{N}$  of polynomial-time functions  $F_a, F_b$ .

$$g_{a,b}(x) = \begin{cases} g(\langle a, b, y' \rangle 0) & \text{if } x = \langle a, b, y' \rangle 0 \\ f(x') & \text{if } x = x' 1 \\ f(0) & \text{else} \end{cases}$$

► **Observation 19.** For  $x = \langle a, b, y' \rangle 0$  it holds that  $g(x) = g_{a,b}(x)$ .

► **Observation 20.** For fixed  $a, b \in \mathbb{N}$  the function  $g_{a,b}(x)$  can be computed in polynomial time.

**Proof.** This follows directly from Observation 18 and the linear runtime of  $f$ . ◀

▷ **Claim 21.** Let  $a \in \mathbb{N}$  and  $y' \in \Sigma^*$  such that  $F_a(y') = \langle i, j, x', 0^p \rangle \notin \overline{D'}$ . Then for all  $b \in \mathbb{N}$  it holds that  $g_{a,b}(\langle a, b, y' \rangle 0) \notin D$ .

**Proof.** By Claim 16 we know that for all  $b \in \mathbb{N}$  the computation  $B(\langle a, b, y' \rangle)$  accepts. This implies  $g_{a,b}(\langle a, b, y' \rangle 0) = \langle 1, b_1, \langle a, b, y' \rangle, 0^{k_B |x|^{9a^2b} + k_B} \rangle \notin D$ . ◀

▷ **Claim 22.** Let  $a \in \mathbb{N}$  such that  $\text{img}(F_a) \subseteq \overline{D'}$ . Then there is some  $b \in \mathbb{N}$  with  $\text{img}(g_{a,b}) = D$ .

**Proof.** Choose  $b \in \mathbb{N}$  according to Claim 17. Then  $D = \text{img}(f) \subseteq \text{img}(g_{a,b})$  because  $f$  is a proof system for  $D$ . Let  $x \in \Sigma^*$ . In the bottom two cases of  $g_{a,b}$  and  $g$  we have  $g_{a,b}(x) \in D$  and  $g(x) \in D$ . So we can assume  $x = \langle a, b, y' \rangle 0$  and  $g_{a,b}(x) = \langle 1, b_1, \langle a, b, y' \rangle, 0^{k_B |x|^{9a^2b} + k_B} \rangle$ . By the choice of  $b$ , it holds  $\langle a, b, y' \rangle \notin L(B)$ . By Observation 15 and the choice of  $k_B$ ,  $B(\langle a, b, y' \rangle)$  runs in time  $k_B |\langle a, b, y' \rangle|^{9a^2b} + k_B \leq k_B |x|^{9a^2b} + k_B$ . Therefore,  $\langle 1, b_1, \langle a, b, y' \rangle, 0^{k_B |x|^{9a^2b} + k_B} \rangle \in D$  and hence  $g_{a,b}(x) \in D$ . This shows  $\text{img}(g_{a,b}) \subseteq D$ . ◀

Finally, we define the p-optimal proof system  $h$  for  $\overline{D'}$ . The key difficulty is that  $h$  wants to output  $F_a(y')$  for all  $a$  and  $y'$  using a short proof only when  $F_a$  is a proof system for  $\overline{D'}$ . To do this  $h$  must be able to check this property efficiently. We can do this as follows: if  $f(F_c(\langle a, b, y' \rangle)) = g_{a,b}(\langle a, b, y' \rangle)$ , then we output  $F_a(y')$  and otherwise some arbitrary word from  $\overline{D'}$ . If  $F_a(y') \notin \overline{D'}$ , we know that there is no  $b \in \mathbb{N}$  such that the corresponding output of  $g_{a,b}$  is in  $D$  and the check correctly fails and  $F_a(y')$  is not outputted. By contraposition we observe that we output  $F_a(y')$  only if it is in  $\overline{D'}$ . Hence,  $h$  is a proof system for  $\overline{D'}$ . Lastly, we show that  $h$  p-simulates all proof systems for  $\overline{D'}$ . Let  $F_a$  be an arbitrary proof system for  $\overline{D'}$ . Then there is a  $b \in \mathbb{N}$  such that  $g_{a,b}$  is a proof system for  $D$ . Let  $c \in \mathbb{N}$  be such that  $f$  p-simulates  $g_{a,b}$  with the function  $F_c$ . So, for all  $y' \in \mathbb{N}$  the function  $h$  outputs  $F_a(y')$  for the input to  $h$  corresponding to  $a, b, c, y'$ . Also this input is short in  $a, b, c, y'$  and can be computed in polynomial time in these parameters.

Let  $h' : \mathbb{N} \rightarrow \mathbb{N}$  be a linear time proof system for  $\overline{D'}$ . We define a function  $h : \mathbb{N} \rightarrow \mathbb{N}$ .

$$h(x) = \begin{cases} \langle i, j, x', 0^p \rangle & \text{if } x = \langle a, b, c, \langle a, b, y' \rangle 0, 0^{k_B \cdot |\langle a, b, y' \rangle|^{9a^2b} + k_B}, 0^{|c| \cdot (|\langle a, b, y' \rangle|^{c+c^2})} \rangle 0 \wedge \\ & f(F_c(\langle a, b, y' \rangle 0)) = g_{a,b}(\langle a, b, y' \rangle 0) \wedge \\ & F_a(y') = \langle i, j, x', 0^p \rangle \\ h'(x') & \text{if } x = x' 1 \\ h'(0) & \text{else} \end{cases}$$

► **Observation 23.**  $h(x)$  runs in time  $O(|x|)$ .

**Proof.** The bottom two cases are trivial. For the first case we observe that checking, whether the input is formatted correctly, can be done in linear time. The part  $0^{k_B \cdot \langle a, b, y' \rangle 0^{9a^2b + k_B}}$  can be checked in linear time by iterated multiplication. The computation  $F_a(y')$  can be simulated in  $O(|a| \cdot (|y'|^a + a) \log(|y'|^a + a)) \subseteq O((|y'|^a + a)^3) \subseteq O(|y'|^{3a}) \subseteq O(|x|)$  time. The computation  $f(F_c(\langle a, b, y' \rangle 0))$  can be simulated in  $O(|c| \cdot (|\langle a, b, y' \rangle 0|^c + c) \log(|\langle a, b, y' \rangle 0|^c + c)) \subseteq O(|x|)$  time and  $g_{a,b}(\langle a, b, y' \rangle 0)$  can be simulated in  $O(|\langle a, b, y' \rangle 0|^{9a^2b}) \subseteq O(|x|)$  time by Observation 18. The output  $\langle i, j, x', 0^p \rangle$  can be computed in  $O(|x|)$  time because all of its elements have been computed in the steps analyzed above. ◀

▷ **Claim 24.**  $h$  is a proof system for  $\overline{D'}$ .

**Proof.** We have  $h \in \text{FP}$  by Observation 23.  $\overline{D'} \subseteq \text{img}(h)$ , since  $\text{img}(h') \subseteq \text{img}(h)$  and  $h'$  is a proof system for  $\overline{D'}$ . We show  $\text{img}(h) \subseteq \overline{D'}$  by contradiction. Assume that there exists  $x \in \Sigma^*$  such that  $h(x) \notin \overline{D'}$ . The last two cases in the definition of  $h$  give values obviously in  $\overline{D'}$ . Thus, we only look at the first case. In particular  $F_a(y') = \langle i, j, x', 0^p \rangle$  and  $g_{a,b}(\langle a, b, y' \rangle 0) = f(F_c(\langle a, b, y' \rangle 0))$ . The second implies directly  $g_{a,b}(\langle a, b, y' \rangle 0) \in \text{img}(f) = \overline{D'}$ . Since  $h(x) = F_a(y')$  in this case, by assumption  $F_a(y') \notin \overline{D'}$ . By Claim 21 we conclude the contradiction  $g_{a,b}(\langle a, b, y' \rangle 0) \notin \overline{D'}$ . ◀

▷ **Claim 25.** Let  $a \in \mathbb{N}$  with  $\text{img}(F_a) \subseteq \overline{D'}$ . Then there exist  $b, c \in \mathbb{N}$ , such that

$$\forall y' \in \Sigma^* : F_a(y') = \langle i, j, x', 0^p \rangle = h(\langle a, b, c, \langle a, b, y' \rangle 0, 0^{k_B \cdot |y'|^{9a^2b + k_B}}, 0^{|c| \cdot (|\langle a, b, y' \rangle 0|^c + c)^2} \rangle 0)$$

**Proof.** Claim 22 shows that there is some  $b \in \mathbb{N}$  such that  $\text{img}(g_{a,b}) = D$ . By Observation 20 this  $g_{a,b}$  is a proof system for  $D$ . Since  $f$  is p-optimal, there exists  $c \in \mathbb{N}$  such that  $f(F_c(x)) = g_{a,b}(x)$  for all  $x \in \Sigma^*$ . Let  $y' \in \Sigma^*$ . From  $\text{img}(F_a) \subseteq \overline{D'}$  it follows  $F_a(y') = \langle i, j, x', 0^p \rangle$  for suitable  $i, j, x', p$ . Hence, in Claim 25 we are always in the first case of  $h$ . It follows

$$h(\langle a, b, c, \langle a, b, y' \rangle 0, 0^{k_B \cdot |y'|^{9a^2b + k_B}}, 0^{|c| \cdot (|\langle a, b, y' \rangle 0|^c + c)^2} \rangle 0) = \langle i, j, x', 0^p \rangle.$$

This shows Claim 25. ◀

Let  $a \in \mathbb{N}$  be arbitrary such that  $F_a$  is a proof system for  $\overline{D'}$ . Choose  $b, c$  according to Claim 25. Then the following  $z : \mathbb{N} \rightarrow \mathbb{N}$  shows  $F_a \leq^p h$ .

$$z(y') = \langle a, b, c, \langle a, b, y' \rangle 0, 0^{k_B \cdot |y'|^{9a^2b + k_B}}, 0^{|c| \cdot (|\langle a, b, y' \rangle 0|^c + c)^2} \rangle 0$$

By Claim 25 it holds  $F_a(y') = h(z(y'))$ . The function  $z$  can be computed in polynomial time, because  $a, b, c \in \mathbb{N}$  and  $k_B \in \mathbb{N}^+$  are constant values for a fixed  $F_a$ . This proves that  $h$  is a p-optimal proof system for  $\overline{D'}$ . ◀

► **Corollary 26.** If DP has p-optimal proof systems, coDP has p-optimal proof systems.

**Proof.** Since  $D \in \text{DP}$ , we obtain that there is a p-optimal proof system for  $D$ . Theorem 7 shows that it follows that there is a p-optimal proof system for  $\overline{D'}$ . The language  $\overline{D'}$  is  $\leq_m^p$ -hard for coDP. By Corollary 3, there are p-optimal proof systems for coDP. ◀

## 4 Oracle Construction

Corollary 26 naturally leads to the question of whether optimal proof systems for DP translate to optimal proof systems for coDP. We show that a proof for this translation cannot be relativizable, i.e., we cannot expect to show this translation with the currently available means. This result is a consequence of the following theorem:

► **Theorem 27.** *There exists an oracle  $O$  with the following properties:*

1.  $\overline{C^O}$  has  $p$ -optimal proof systems (implying  $p$ -optimal proof systems for  $\text{coNP}$ ).
2.  $\overline{D^O}$  has no optimal proof systems (ruling out optimal proof systems for  $\text{coDP}$ ).

**Sketch of the construction.** Work towards 1 (coding): For all  $a \in \mathbb{N}$ , the construction tries to achieve that  $\text{img}(F_a^O) \neq \overline{C^O}$  and thus,  $F_a^O$  is no proof system for  $\overline{C^O}$ . If this is not possible, we start to *encode* the mappings of  $F_a$  (i.e., on which input it gives which output) into the oracle. For the coding, we define code words of the form  $c(a, x, y) := \langle 0^a, 0^{4(|x|^a + a + |y|)}, x, y \rangle$  for  $a \in \mathbb{N}$ ,  $x, y \in \Sigma^*$ . The purpose of a code word  $c(a, x, y)$  is to encode the computation  $F_a(x) = y$ . Thus, the final oracle  $O$  will contain the encoded mappings of all proof systems for  $\overline{C^O}$ . The crucial point is that such a code word lets us recognize that  $F_a$  is a proof system for  $\overline{C^O}$  and  $y \in \overline{C^O}$ . This allows us to define a  $p$ -optimal proof system  $h$  which is able to simulate every proof system for  $\overline{C^O}$  using oracle queries.

Work towards 2 (diagonalization): For all  $b \in \mathbb{N}$ , the construction tries to achieve that  $\text{img}(G_b^O) \neq \overline{D^O}$  and thus,  $G_b^O$  is no proof system for  $\overline{D^O}$ . If this is not possible, we define some proof system  $z_b^O$  for  $\overline{D^O}$  and show that  $z_b^O$  cannot be simulated by  $G_b^O$ . The latter is achieved by diagonalizing against every simulation function  $\pi$ , i.e., we make sure that  $G_b^O$  does not simulate  $z_b^O$  via  $\pi$ .

We call the functions  $z_b^O$  witness proof systems. The intuition behind their definition and behind the whole diagonalization is as follows: independent of the oracle each function  $z_b^O$  has short proofs for all elements of some polynomial-time-decidable set. But our construction offers the freedom to choose whether or not this set is a subset of  $\overline{D^O}$ . The latter depends on the following language

$$\begin{aligned} A^O &= \{x \in \Sigma^* \mid |O^{|x|}|_c \geq 2\} \cup \{x \in \Sigma^* \mid |O^{|x|}|_c = 0\} \\ &= \{x \in \Sigma^* \mid |O^{|x|}|_c \neq 1\}, \end{aligned}$$

which lies inside  $\text{coDP}^O$  and thus has influence on  $\overline{D^O}$ . Let  $y$  be a word whose membership to  $\overline{D^O}$  is influenced by the question of whether  $0^n \in A^O$ . Observe that  $0^n \in A^O$  if and only if  $|O|_c^n \neq 1$ . Thus, we can control the membership of  $y$  to  $\overline{D^O}$  by adding none, one or more words of length  $n$  to  $O$ . There are  $2^n$  such words. Let  $G_b^O$  be some proof system for  $\overline{D^O}$ . During the construction of  $O$ , we initially have no word of length  $n$  inside  $O$  and thus  $y \in \overline{D^O}$  and  $G_b^O$  must have a proof for  $y$ . Case 1: All  $G_b^O$ -proofs for  $y$  are long. When  $G_b^O$  is given such a proof it can determine by exhaustive search the number of words of length  $n$  in  $O$ . However,  $G_b^O$  does not simulate  $z_b^O$ , because  $z_b^O$  has short proofs for  $y$ , but  $G_b^O$  has not. Case 2:  $G_b^O$  has a short proof  $x$  for  $y$ . In this case,  $G_b^O(x)$  cannot query all  $2^n$  words and hence cannot determine whether  $y \in \overline{D^O}$ . We can exploit this to create a situation where  $G_b^O(x)$  outputs an element outside  $\overline{D^O}$  and hence is no proof system for this set. So  $G_b^O$  can either simulate  $z_b^O$  or be a proof system for  $\overline{D^O}$ , but not both at once.

The main challenge of the oracle construction is to combine the work for 1 and 2, because the code words interact with the diagonalization. Indeed, in the example above  $G_b^O$  cannot query all  $2^n$  words when having a short proof  $x$  for  $y$ , but there are many code words that can be queried by  $G_b^O(x)$  whose memberships together can depend on all  $2^n$  words of length  $n$ . We capture these dependencies in a graph data structure, where nodes are words from  $\Sigma^*$  and edges are oracle queries of underlying FP-computations of code words, i.e., for a code word  $c(a, x, y)$  the computation  $F_a^O(x)$ . This helps to identify words of length  $n$  that are independent of the computation  $G_b^O(x)$  and all queried code words.

## 5 Conclusion

We summarize all results to obtain the equivalence classes from Figure 1. First observe that (p-)optimal proof systems always translate from a class  $\mathcal{C}$  to  $\mathcal{D}$  when  $\mathcal{C} \subseteq \mathcal{D}$  (respective solid arrows are omitted in Figure 2). We start with the equivalence classes for p-optimal proof systems (see Figure 2, left, solid arrows). P-optimal proof systems translate as follows:

- from  $\text{NP} \cup \text{coNP}$  to  $\text{DP}$  by  $\text{NP} \cup \text{coNP} \supseteq \text{NP}$ ,  $\text{NP} \cup \text{coNP} \supseteq \text{coNP}$ ,  $\text{NP} \wedge \text{coNP} = \text{DP}$ , and Corollary 5 following from Köbler, Messner, and Torán [18].
- from  $\text{DP}$  to  $\text{coDP}$  by Corollary 26.
- from  $\text{coBH}_k$  to  $\text{coBH}_{k+1}$  for  $k \geq 2$  by Corollary 5 following from Köbler, Messner, and Torán [18] and the following inclusions:

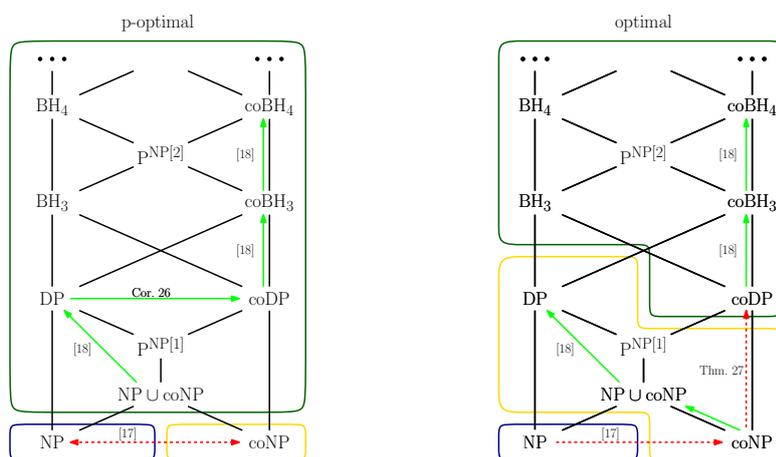
$$\begin{aligned} \text{coBH}_k \wedge \text{coBH}_k &\supseteq \text{coDP} \wedge \text{coBH}_k = (\text{NP} \vee \text{coNP}) \wedge \text{coBH}_k \\ &\supseteq \begin{cases} \text{coNP} \wedge \text{coBH}_k = \text{coBH}_{k+1} & \text{if } k \text{ is even} \\ \text{NP} \vee \text{coBH}_k = \text{coBH}_{k+1} & \text{else} \end{cases} \end{aligned}$$

Next, we derive the equivalence classes for optimal proof systems (see Figure 2, right, solid arrows). Optimal proof systems translate as follows:

- from  $\text{coNP}$  to  $\text{NP} \cup \text{coNP}$  by the fact that  $\text{NP}$  has optimal proof systems.
- from  $\text{NP} \cup \text{coNP}$  to  $\text{DP}$  by Corollary 5 following from Köbler, Messner, and Torán [18].
- from  $\text{coBH}_k$  to  $\text{coBH}_{k+1}$  for  $k \geq 2$  by the same argument used for p-optimal proof systems.

The resulting equivalence classes for (p-)optimal proof systems are different (see Figure 2, left and right, dashed arrows) relative to oracles  $A, B, O$  with the following properties:

- $\text{NP}^A$  has p-optimal proof systems and  $\text{coNP}^A$  has no optimal proof systems [17].
- $\text{coNP}^B$  has p-optimal proof systems and  $\text{NP}^B$  has no p-optimal proof systems [17].
- $\text{coNP}^O$  has p-optimal proof systems and  $\text{coDP}^O$  has no optimal proof systems (Thm. 27).



■ **Figure 2** Equivalence classes for p-optimal proof systems (left) and optimal proof systems (right) in the Boolean hierarchy over  $\text{NP}$  and the bounded query hierarchy over  $\text{NP}$ . Green solid arrows from  $A$  to  $B$  mean that (p-)optimal proof systems for  $A$  imply (p-)optimal proof systems for  $B$ . Red dashed arrows from  $A$  to  $B$  mean that there exists an oracle  $Q$  relative to which  $A^Q$  has (p-)optimal proof systems and  $B^Q$  has no (p-)optimal proof systems. Note that green solid arrows pointing downwards are omitted, since those are trivial and only the minimum number of required red dashed arrows to separate all equivalence classes are drawn.

Oracle  $A$  rules out translations from NP to any other class in Figure 2 for optimal and p-optimal proof systems. Oracle  $B$  rules out translations from coNP to NP and thus also to  $\text{NP} \cup \text{coNP}$  for p-optimal proof systems. Oracle  $O$  rules out translations from coNP to coDP for optimal proof systems.

We obtain the following connection to a conjecture studied by Pudlák [25].

► **Corollary 28.** *The following statements are equivalent:*

- BH has no p-optimal proof system.
- TAUT has no p-optimal proof systems or SAT has no p-optimal proof systems (i.e.,  $\text{CON} \vee \text{SAT}$  in Pudlák's notation).

**Proof.** Figure 2 shows that  $\text{NP} \cup \text{coNP}$  and BH are equivalent with respect to p-optimal proof systems. Hence, BH has no p-optimal proof systems if and only if  $\text{NP} \cup \text{coNP}$  has no p-optimal proof systems. The latter holds if and only if TAUT has no p-optimal proof systems or SAT has no p-optimal proof systems. ◀

---

## References

- 1 T. P. Baker, J. Gill, and R. Solovay. Relativizations of the  $\text{P} = ? \text{NP}$  question. *SIAM Journal on Computing*, 4(4):431–442, 1975. doi:10.1137/0204037.
- 2 Richard Beigel. Bounded queries to sat and the boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, 1991. doi:10.1016/0304-3975(91)90160-4.
- 3 O. Beyersdorff. Representable disjoint NP-pairs. In *Proceedings 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2004. doi:10.1007/978-3-540-30538-5\_11.
- 4 O. Beyersdorff. Disjoint NP-pairs from propositional proof systems. In *Proceedings of Third International Conference on Theory and Applications of Models of Computation*, volume 3959 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2006. doi:10.18452/15520.
- 5 O. Beyersdorff. Classes of representable disjoint NP-pairs. *Theoretical Computer Science*, 377(1-3):93–109, 2007. doi:10.1016/j.tcs.2007.02.005.
- 6 O. Beyersdorff. The deduction theorem for strong propositional proof systems. *Theory of Computing Systems*, 47(1):162–178, 2010. doi:10.1007/s00224-008-9146-6.
- 7 O. Beyersdorff, J. Köbler, and J. Messner. Nondeterministic functions and the existence of optimal proof systems. *Theoretical Computer Science*, 410(38-40):3839–3855, 2009. doi:10.1016/j.tcs.2009.05.021.
- 8 O. Beyersdorff and Z. Sadowski. Do there exist complete sets for promise classes? *Mathematical Logic Quarterly*, 57(6):535–550, 2011. doi:10.1002/malq.201010021.
- 9 S. R. Buss. Lectures on proof theory. Technical Report SOCS 96.1, McGill University, 1996.
- 10 S. Cook and J. Krajíček. Consequences of the provability of  $\text{NP} \subseteq \text{P/poly}$ . *Journal of Symbolic Logic*, 72(4):1353–1371, 2007. doi:10.2178/jsl/1203350791.
- 11 S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979. doi:10.2307/2273702.
- 12 M. I. Dekhtyar. On the relativization of deterministic and nondeterministic complexity classes. In *Mathematical Foundations of Computer Science*, volume 45 of *Lecture Notes in Computer Science*, pages 255–259, Berlin, 1976. Springer-Verlag.
- 13 C. Glaßer, A. L. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370:60–73, 2007. doi:10.1007/11549345\_35.
- 14 C. Glaßer, A. L. Selman, and L. Zhang. The informational content of canonical disjoint NP-pairs. *International Journal of Foundations of Computer Science*, 20(3):501–522, 2009. doi:10.1007/978-3-540-73545-8\_31.

- 15 E. A. Hirsch. Optimal acceptors and optimal proof system. In J. Kratochvíl, A. Li, J. Fiala, and P. Kolman, editors, *Theory and Applications of Models of Computation*, pages 28–39. Springer Berlin Heidelberg, 2010.
- 16 E. A. Hirsch and D. Itsykson. On optimal heuristic randomized semidecision procedures, with application to proof complexity. In J. Marion and T. Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science*, volume 5, pages 453–464. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010. doi:10.4230/LIPIcs.STACS.2010.2475.
- 17 E. Khaniki. New relations and separations of conjectures about incompleteness in the finite domain. *The Journal of Symbolic Logic*, 87(3):912–937, 2022. doi:10.1017/js1.2021.99.
- 18 J. Köbler, J. Messner, and J. Torán. Optimal proof systems imply complete sets for promise classes. *Information and Computation*, 184(1):71–92, 2003. doi:10.1016/S0890-5401(03)00058-0.
- 19 J. Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CB09780511529948.
- 20 J. Krajíček. *Proof Complexity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2019. doi:10.1017/9781108242066.
- 21 J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *Journal of Symbolic Logic*, 54:1063–1079, 1989. doi:10.2307/2274765.
- 22 J. Messner. *On the Simulation Order of Proof Systems*. PhD thesis, Universität Ulm, 2000.
- 23 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.
- 24 P. Pudlák. On the lengths of proofs of consistency. In *Collegium Logicum*, pages 65–86. Springer Vienna, 1996. doi:10.1016/S0049-237X(08)70462-2.
- 25 P. Pudlák. Incompleteness in the finite domain. *The Bulletin of Symbolic Logic*, 23(4):405–441, 2017.
- 26 P. Pudlák. The lengths of proofs. In Samuel R. Buss, editor, *Handbook of Proof Theory*, pages 547–637. Elsevier, Amsterdam, 1998.
- 27 A. Razborov. On provably disjoint NP-pairs. *BRICS Report Series*, 36, 1994. doi:10.7146/brics.v1i36.21607.
- 28 Z. Sadowski. On an optimal propositional proof system and the structure of easy subsets of TAUT. *Theoretical Computer Science*, 288(1):181–193, 2002. doi:10.1016/S0304-3975(01)00155-4.
- 29 T. Pitassi and R. Santhanam. Effectively polynomial simulations. In *Innovations in Computer Science*, pages 370–381, 2010.
- 30 K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1):53–80, 1987. doi:10.1016/0304-3975(87)90049-1.



# Finding a Highly Connected Steiner Subgraph and its Applications

Eduard Eiben  

Royal Holloway, University of London, Egham, UK

Diptapriyo Majumdar<sup>1</sup>  

Indraprastha Institute of Information Technology Delhi, New Delhi, India

M. S. Ramanujan  

University of Warwick, Coventry, UK

---

## Abstract

---

Given a (connected) undirected graph  $G$ , a set  $X \subseteq V(G)$  and integers  $k$  and  $p$ , the STEINER SUBGRAPH EXTENSION problem asks whether there exists a set  $S \supseteq X$  of at most  $k$  vertices such that  $G[S]$  is a  $p$ -edge-connected subgraph. This problem is a natural generalization of the well-studied STEINER TREE problem (set  $p = 1$  and  $X$  to be the terminals). In this paper, we initiate the study of STEINER SUBGRAPH EXTENSION from the perspective of parameterized complexity and give a fixed-parameter algorithm (i.e., FPT algorithm) parameterized by  $k$  and  $p$  on graphs of bounded degeneracy (removing the assumption of bounded degeneracy results in W-hardness).

Besides being an independent advance on the parameterized complexity of network design problems, our result has natural applications. In particular, we use our result to obtain new single-exponential FPT algorithms for several vertex-deletion problems studied in the literature, where the goal is to delete a smallest set of vertices such that: (i) the resulting graph belongs to a specified hereditary graph class, and (ii) the deleted set of vertices induces a  $p$ -edge-connected subgraph of the input graph.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms; Theory of computation  $\rightarrow$  Dynamic programming

**Keywords and phrases** Parameterized Complexity, Steiner Subgraph Extension,  $p$ -edge-connected graphs, Matroids, Representative Families

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.45

**Funding** *M. S. Ramanujan*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grants EP/V007793/1 and EP/V044621/1.

## 1 Introduction

Given a simple undirected graph  $G = (V, E)$  and a set  $T \subseteq V(G)$ , called *terminals*, the STEINER TREE problem asks if there are at most  $k$  edges  $F \subseteq E(G)$  such that there is a path between every pair of vertices of  $T$  in  $G' = (V, F)$ . STEINER TREE is one of the fundamental problems in network design and is a well-studied problem in parameterized complexity ([14, 9, 7, 3, 21]). We refer to Section 2 for definitions related to parameterized complexity and graph theory. In this paper, we study the STEINER SUBGRAPH EXTENSION problem, which is formally defined below.

STEINER SUBGRAPH EXTENSION

**Input:** A simple undirected graph  $G = (V, E)$ ,  $X \subseteq V(G)$  and integers  $k, p \in \mathbb{N}$ .

**Parameter:**  $k + p$

**Goal:** Is there  $S \supseteq X$  of size at most  $k$  such that  $G[S]$  is  $p$ -edge-connected?

---

<sup>1</sup> Corresponding author



Observe that this is a natural generalization of STEINER TREE problem. To the best of our knowledge, the parameterized complexity status of STEINER SUBGRAPH EXTENSION is unexplored even for arbitrary fixed constant  $p$ . When  $p = 2$ , STEINER SUBGRAPH EXTENSION is closely related to a special variant of EDGE-CONNECTED SURVIVABLE NETWORK DESIGN (defined by Feldman et al. [15]) problem. The goal of EDGE-CONNECTED SURVIVABLE NETWORK DESIGN is to find a collection of “at most  $k$  edges” so that there are two edge-disjoint paths between every pair of vertices in the terminal set. Abhinav et al. [1] studied the above problem when  $p = n - k$ , with  $k$  as the parameter. Moreover, they aim to find an  $(n - k)$ -edge-connected steiner subgraph with exactly  $\ell$  vertices. In our problem, observe that  $p \leq k - 1$  as any graph with  $k$  vertices can be  $(k - 1)$ -edge-connected and not  $p$ -edge-connected for  $p \geq k$ . If we set  $p = k - 1$ , then our problem becomes precisely the CLIQUE problem, where we want to decide if a graph has a clique with exactly  $k$  vertices, a W[1]-hard problem. Hence, one must place further restrictions on the input when aiming for fixed-parameter tractability. In this paper, we consider the STEINER SUBGRAPH EXTENSION problem when  $\eta$  is the degeneracy of the input graph  $G$  and  $\eta$  is a fixed-constant. Note that many well-known sparse graph classes are subclasses of graphs of bounded degeneracy. For instance, planar graphs are 5-degenerate, graphs with treewidth (or treedepth or pathwidth) at most  $\eta$  are  $(\eta + 1)$ -degenerate.

**Our Contributions.** The input to our problem STEINER SUBGRAPH EXTENSION is a simple undirected graph with  $n$  vertices and  $\eta$  is a fixed constant. Recall that the parameter is  $k + p$ . The first part of our paper proves that STEINER SUBGRAPH EXTENSION is FPT when the input graph has constant degeneracy. In particular, we give an FPT algorithm with running time  $2^{\mathcal{O}(pk+\eta)}n^{\mathcal{O}(1)}$ -time for STEINER SUBGRAPH EXTENSION when the input graph  $G$  is  $\eta$ -degenerate. The formal statement of the theorem is given below.

► **Theorem 1.** STEINER SUBGRAPH EXTENSION can be solved in time  $2^{\mathcal{O}(pk+\eta)}n^{\mathcal{O}(1)}$ , where  $\eta$  is the degeneracy of the input graph.

In particular, on graphs of constant degeneracy and for constant  $p$ , the above result gives a  $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ -time algorithm, which is useful in several applications as we show in this paper.

The above result crucially relies on the use of the *out-partition* matroid, its linear representability in deterministic polynomial-time, and a dynamic programming subroutine using the notion of representative sets. We would like to highlight that Einarson et al. [13] have studied the same problem when  $X$  is a vertex cover. Our dynamic programming algorithm over representative sets has some similarities with the algorithm of Einarson et al. [13] but  $X$  is not necessarily a vertex cover of  $G$  for STEINER SUBGRAPH EXTENSION. Despite the fact that  $G$  is a bounded degenerate graph, designing an algorithm for STEINER SUBGRAPH EXTENSION needs careful adjustment to the subproblem definitions and some additional conditions have to be incorporated while constructing the collection of sets in the DP formulation. Furthermore, our algorithm in Theorem 1 does not depend on  $\eta$  in the exponent of  $n$ .

The second part of our paper describes some applications of our main result (Theorem 1) to some natural problems in graph theory with connectivity constraints. Einarson et al. [13] have initiated the study of  $p$ -EDGE-CONNECTED VERTEX COVER with stronger connectivity constraints. Being motivated by their results, we illustrate how Theorem 1 lays us a foundation to *design efficient deterministic parameterized singly exponential-time algorithms* for BOUNDED DEGREE DELETION SET,  $\eta$ -TREEDPTH DELETION SET, PATHWIDTH-ONE DELETION SET and  $\eta$ -PATH VERTEX COVER with  $p$ -edge-connectivity constraints. Each of

these problems are well-studied without the connectivity constraints (see [18, 5, 6] for more details). We state the problem definitions below. Given an undirected graph  $G = (V, E)$ , the following questions are asked by these problems.

- $p$ -EDGE-CONNECTED  $\eta$ -DEGREE DELETION SET ( $p$ -EDGE-CON-BDDS) asks if there is  $S \subseteq V(G)$  such that  $G - S$  is a graph of maximum degree at most  $\eta$  and  $G[S]$  is  $p$ -edge-connected.
- $p$ -EDGE-CONNECTED  $\eta$ -TREEDEPTH DELETION SET ( $p$ -EDGE-CON- $\eta$ -TDDS) asks if there is  $S \subseteq V(G)$  such that  $G - S$  has treedepth at most  $\eta$  and  $G[S]$  is  $p$ -edge-connected.
- $p$ -EDGE CONNECTED PATHWIDTH-1 VERTEX DELETION ( $p$ -EDGE-CON-PW1DS) asks if there is  $S \subseteq V(G)$  such that  $G - S$  has pathwidth at most 1 and  $G[S]$  is  $p$ -edge-connected.
- $p$ -EDGE-CONNECTED  $\eta$ -PATH VERTEX COVER ( $p$ -EDGE-CON- $\eta$ -PVC) asks if there is  $S \subseteq V(G)$  such that  $G - S$  has no  $P_\eta$  as subgraph and  $G[S]$  is  $p$ -edge-connected.

Applications to each of the above mentioned problems crucially rely on a property. The property is that all minimal vertex-deletion sets that must be part of any optimal solution can be enumerated in  $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ -time for some fixed constant  $\eta$ . Since a graph of maximum degree  $\eta$  is also an  $\eta$ -degenerate graph, we have the following result as a direct application of our main result.

► **Corollary 2.**  $p$ -EDGE-CON-BDDS admits a  $2^{\mathcal{O}(pk+k\eta)}n^{\mathcal{O}(1)}$ -time algorithm.

Our second application is  $p$ -EDGE-CON-PW1DS problem. The graphs of pathwidth at most one are also 2-degenerate. But it is not straightforward to enumerate all the minimal pathwidth one vertex deletion sets. So we use some additional characterizations of graphs of pathwidth one and exploit some problem specific structures to prove our next result.

► **Theorem 3.**  $p$ -EDGE-CON-PW1DS admits an algorithm that runs in  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time.

Note that the algorithm for the above result does not directly invoke the subroutine from Theorem 1. Instead, it uses some dynamic programming ideas that are closely similar to that of Theorem 1 proof but also makes careful local adjustments to take care of some additional constraints. Finally, our last two applications are  $p$ -EDGE-CONNECTED  $\eta$ -TREEDEPTH DELETION SET and  $p$ -EDGE-CON- $\eta$ -PVC problems and we have the following two results.

► **Theorem 4.**  $p$ -EDGE-CON- $\eta$ -TDDS admits an algorithm that runs in  $2^{2^{2^\eta} + \mathcal{O}((p+\eta)k)}n^{2^{2^\eta}}$ -time.

► **Theorem 5.**  $p$ -EDGE-CON- $\eta$ -PVC admits an algorithm that runs in  $2^{\mathcal{O}((p+\eta)k)}n^{\mathcal{O}(1)}$ -time.

**Organization of our paper.** We organize the paper as follows. Initially in Section 2, we introduce the basic notations related to graph theory, parameterized complexity and matroids. Then, in Section 3, we prove our main result, i.e. Theorem 1. Then, in Section 4, we illustrate the applications of our main result to design singly exponential-time algorithms for  $p$ -EDGE-CON-BDDS,  $p$ -EDGE-CON-PW1DS,  $p$ -EDGE-CON- $\eta$ -TDDS and  $p$ -EDGE-CON- $\eta$ -PVC.

**Related Work.** Heggernes et al. [19] studied the parameterized complexity of  $p$ -CONNECTED STEINER SUBGRAPH that is the vertex-connectivity counterpart of our problem. The authors in their paper have proved that when parameterized by  $k$ , the above mentioned problem is FPT for  $p = 2$  and W[1]-hard when  $p = 3$ . Nutov [23] has studied a variant of  $p$ -CONNECTED STEINER SUBGRAPH problem in which they have studied VERTEX CONNECTIVITY AUGMENTATION problem. Given an undirected graph  $G$ , a  $p$ -connected subgraph  $G[S]$ , the

VERTEX CONNECTIVITY AUGMENTATION problem asks if at most  $k$  additional edges can be added to  $G[S]$  to make the subgraph  $(p+1)$ -connected. In particular, Nutov [23] provided a parameterized algorithm for the above mentioned problem. Feldman et al. [15] have studied parameterized complexity of VERTEX/EDGE-CONNECTED SURVIVABLE NETWORK DESIGN PROBLEM where given fixed constant  $p$ , they want to compute a subgraph that has minimum number of edges and provides  $p$ -vertex/edge-connectivity between every pair of vertices in the terminals.

## 2 Preliminaries

**Sets, numbers and graph theory.** We use  $\mathbb{N}$  to denote the set of all natural numbers. For  $r \in \mathbb{N}$ , we use  $[r]$  to denote the set  $\{1, \dots, r\}$ . Given a set  $S$  and an integer  $k$ , we use  $\binom{S}{\leq k}$  and  $\binom{S}{k}$  to denote the collection of all subsets of  $S$  of size at most  $k$  and of size exactly  $k$  respectively. We use standard graph theoretic notations from Diestel's book [10] for all notations of undirected and directed graphs. For undirected graphs, we use  $uv \in E(G)$  to denote that there is an edge between  $u$  and  $v$ . On the other hand for directed graphs, we are more explicit. We use  $(u, v)$  to represent that the edge is directed from  $u$  to  $v$ . For the directed graphs, the directed edges are also called arcs. We use the term arc and edge interchangeably. In an undirected graph  $G$ , we use  $\deg_G(v)$  to denote the degree of  $v$  in  $G$ . When the graph is clear from the context, we drop this subscript and simply use  $\deg(v)$ . An undirected graph  $G$  is called a *degree- $\eta$ -graph* if every vertex of  $G$  has degree at most  $\eta$ . We use  $\Delta(G)$  to denote the  $\max\{\deg_G(v) : v \in V(G)\}$ , i.e. the maximum degree of any vertex in  $G$ . It is clear from the definition that if a graph  $G$  is a degree- $\eta$ -graph then  $\Delta(G) \leq \eta$ . When we consider directed graphs, we have *in-degree* and *out-degree* for all the vertices. For a vertex  $v$ , the *in-degree* of  $v$  is the number of arcs of the form  $(v, u) \in A$  and the *out-degree* of  $v$  is the number of arcs of the form  $(u, v) \in A$ . A connected undirected graph  $G = (V, E)$  is said to be  *$p$ -edge-connected* if at least two vertices and  $G - Y$  remains connected after deleting at most  $p - 1$  edges. Due to the Menger's Theorem, a connected graph is said to be  *$p$ -edge-connected* if and only if there are  $p$  edge-disjoint paths between every pair of vertices. Given an undirected graph, a set  $S \subseteq V(G)$  is said to be a  *$p$ -segment* of  $G$  if for every  $u, v \in S$ , there are  $p$  edge-disjoint paths from  $u$  to  $v$  in  $G$ . An undirected graph is said to be an  *$\eta$ -degenerate graph* if every subgraph has a vertex of degree at most  $\eta$ . Given an undirected  $\eta$ -degenerate graph  $G = (V, E)$ , a sequence of vertices  $\rho_G = (v_1, \dots, v_n)$  is said to be an  *$\eta$ -degeneracy sequence* if for every  $2 \leq i \leq n$ ,  $v_i$  has at most  $\eta$  neighbors from the vertices  $\{v_1, \dots, v_{i-1}\}$ . For an  $\ell \in \mathbb{N}$ , we use  $P_\ell$  to denote a path containing  $\ell$  vertices and  $C_\ell$  to denote a cycle containing  $\ell$  vertices. A graph is said to be a *degree- $\eta$ -graph* if every vertex has degree at most  $\eta$ . It follows from the definition that every degree- $\eta$ -graph is an  $\eta$ -degenerate graph, but the converse does not hold true. An undirected graph is said to be a *caterpillar graph* if every connected component is an induced path with hairs attached to each of its pendant vertices. Given a directed graph  $D = (V, A)$ , we define an *outbranching* of  $D$  rooted at  $v \in V(D)$  is a subset  $A' \subseteq A$  such that  $v$  has in-degree 0 and every other vertex has in-degree exactly one in  $D' = (V, A')$ .

We define the following two graph parameters treedepth and pathwidth that we use in our paper.

► **Definition 6 (Treedepth).** Given an undirected graph  $G = (V, E)$ ,  $\text{td}(G)$ , i.e. the treedepth of  $G$  is defined as follows. If  $|V(G)| = 1$ , then  $\text{td}(G) = 1$ . If  $G$  is connected, then  $\text{td}(G) = 1 + \min_{u \in V(G)} \text{td}(G - \{u\})$ . Finally, if  $G_1, \dots, G_r$  are the connected components of  $G$ , then  $\text{td}(G) = \max_{i=1}^r \text{td}(G_i)$ .

Informally, a *treedepth decomposition* of an undirected graph  $G$  can be considered as a rooted forest  $Y$  with vertex set  $V$  such that for each  $uv \in E(G)$ , either  $u$  is an ancestor of  $v$  or  $v$  is an ancestor of  $u$  in  $Y$ . The context of treedepth is also sometimes referred to as *elimination tree* of  $G$ . It follows from the (recursive) definition above that treedepth of a graph is referred to as the minimum depth of a treedepth decomposition of  $G$ , where depth is defined as the maximum number of vertices in a root to leaf path.

► **Definition 7 (Path Decomposition).** A path decomposition of an undirected graph  $G = (V, E)$  is a sequence  $(X_1, \dots, X_r)$  of bags  $X_i \subseteq V(G)$  such that **(i)** every vertex belongs to at least one bag, **(ii)** for every edge  $uv \in E(G)$ , there is  $X_i$  such that  $u, v \in X_i$ , and **(iii)** for every vertex  $v$ , the bags containing  $v$  forms a contiguous subsequence, i.e.  $(X_i, X_{i+1}, \dots, X_j)$ . The width of a decomposition is  $\max_{i \in [r]} |X_i| - 1$ .

The *pathwidth* of a graph is defined as the smallest number  $\eta$  such that there exists a path decomposition of width  $\eta$ . Informally, pathwidth is a measure how much a graph is close to a path (or a linear forest). We use  $\text{pw}(G)$  to denote the pathwidth of  $G$ .

Given a class of graphs  $\mathcal{G}$ , we say that  $\mathcal{G}$  is *polynomial-time recognizable*, if given a graph  $G$ , there is a polynomial-time algorithm that can correctly check if  $G \in \mathcal{G}$ . A graph class is said to be *hereditary* if it is closed under induced subgraphs.

**Parameterized Complexity and W-hardness.** A parameterized problem  $L$  is a subset of  $\Sigma^* \times \mathbb{N}$  for some finite alphabet  $\Sigma$ . An instance of a parameterized problem is a pair  $(x, k)$  where  $x \in \Sigma^*$  is the *input* and  $k$  is the *parameter*. A parameterized problem  $L \subseteq \Sigma^* \times \mathbb{N}$  is said to be *fixed-parameter tractable* if there exists an algorithm  $\mathcal{A}$  that given  $(x, k) \in \Sigma^* \times \mathbb{N}$ , the algorithm  $\mathcal{A}$  runs in  $f(k)|x|^c$ -time for some constant  $c$  independent of  $n$  and  $k$  and correctly decides  $L$ . The algorithm  $\mathcal{A}$  that runs in  $f(k)|x|^{\mathcal{O}(1)}$ -time is called a *fixed-parameter algorithm* (or FPT algorithm). A fixed-parameter algorithm is said to be a *singly exponential FPT algorithm* if it runs in  $c^k|x|^{\mathcal{O}(1)}$ -time for some fixed constant  $c$  independent of  $|x|$  and  $k$ . There is a hardness theory in parameterized complexity that is associated with the notion of *parameterized reduction* and the hierarchy of parameterized complexity classes. Broadly, the *W-hierarchy* (of parameterized complexity classes) is denoted by  $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \dots \subseteq \text{XP}$ . Given two distinct parameterized problems  $L_1$  and  $L_2$ , there is a *parameterized reduction* from  $L_1$  to  $L_2$  if given an instance  $(x, k)$  of  $L_1$ , an algorithm  $\mathcal{A}$  runs in  $g(k)|x|^{\mathcal{O}(1)}$ -time and outputs an equivalent instance  $(x', k')$  of  $L_2$  such that  $k' = f(k)$  for some function depending only on  $k$ . For more details on parameterized complexity and its associated hardness theory, we refer to the books [8, 22, 11].

**Matroids and Representative Families.** We use the following definitions and results related to matroid theory to design our algorithms.

► **Definition 8.** Given a universe  $U$  and a subfamily  $\mathcal{I} \subseteq 2^U$ , a set system  $\mathcal{M} = (U, \mathcal{I})$  is said to be a matroid if **(i)**  $\emptyset \in \mathcal{I}$ , **(ii)** if  $A \in \mathcal{I}$ , then for all  $A' \subseteq A$ ,  $A' \in \mathcal{I}$  (*hereditary property*), and **(iii)** if there exists  $A, B \in \mathcal{I}$  such that  $|B| > |A|$ , then there is  $x \in B \setminus A$  such that  $A \cup \{x\} \in \mathcal{I}$  (*exchange property*). The set  $U$  is called ground set of  $\mathcal{M}$  and a set  $A \in \mathcal{I}$  is called an independent set of matroid  $\mathcal{M}$ .

It follows from the definition that all maximal independent sets are of the same size. A maximal independent set is called a *basis*. Let  $U$  be a universe with  $n$  elements and  $\mathcal{I} = \binom{U}{\leq r}$ . The set system  $(U, \mathcal{I})$  is called a *uniform matroid*. Let  $G = (V, E)$  be an undirected graph and  $\mathcal{I} = \{F \subseteq E(G) \mid G' = (V, F) \text{ is a forest}\}$ . The set system  $(E(G), \mathcal{I})$  is called a *graphic matroid*. Let  $U$  be partitioned as  $U_1 \uplus \dots \uplus U_r$  and  $\mathcal{I} = \{A \subseteq U : |A \cap U_i| \leq 1 \text{ for all } i \in [r]\}$ .

We say that  $(U, \mathcal{I})$  is a *partition matroid*. A matroid  $M$  is said to be *representable over a field*  $\mathbb{F}$  if there is a matrix  $A$  over  $\mathbb{F}$  and a bijection  $f : U \rightarrow \text{Col}(A)$  where  $\text{Col}(A)$  is the set of columns of  $A$  such that  $B \subseteq U$  is an independent set of  $U$  if and only if the set of columns  $\{f(b) \mid b \in B\}$  are linearly independent. A matroid representable over a field  $\mathbb{F}$  is called a *linear matroid*.

Given two matroids  $M_1 = (U_1, \mathcal{I}_1)$  and  $M_2 = (U_2, \mathcal{I}_2)$ , the *direct sum*  $M = M_1 \oplus M_2$  is the matroid  $(U_1 \uplus U_2, \mathcal{I})$  such that  $I \in \mathcal{I}$  if and only if  $I \cap U_1 \in \mathcal{I}_1$  and  $I \cap U_2 \in \mathcal{I}_2$ . If  $M_1$  and  $M_2$  are represented by matrices  $A_1$  and  $A_2$  respectively then  $M = M_1 \oplus M_2$  also admits a matrix representation.

Given a matroid  $M$ , a *truncation* of  $M$  to rank  $r$  is the matroid  $M' = (U, \mathcal{I}')$  where a set  $A \subseteq U$  is independent in  $M'$  if and only if  $A \in \mathcal{I}$  and  $|A| \leq r$ . Given a matroid  $M$  with its representation (in matrix-form), the truncation of  $M$  can be computed in polynomial-time. Let  $M = (U, \mathcal{I})$  be a matroid and  $X, Y \subseteq U$ . We say that  $X$  *extends*  $Y$  in  $M$  if  $X \cap Y = \emptyset$  and  $X \cup Y \in \mathcal{I}$ . Moreover, let  $\mathcal{S} \subseteq 2^U$  be a family. A subfamily  $\hat{\mathcal{S}} \subseteq \mathcal{S}$  is a *q-representative* of  $\mathcal{S}$  if the following holds: for every set  $Y \subseteq U$  with  $|Y| \leq q$ , there is a set  $X \in \hat{\mathcal{S}}$  such that  $X$  extends  $Y$  if and only if there is a set  $\hat{X} \in \hat{\mathcal{S}}$  such that  $\hat{X}$  extends  $Y$ . We use  $\hat{\mathcal{S}} \subseteq_{\text{rep}}^q \mathcal{S}$  to denote that  $\hat{\mathcal{S}}$  is a *q-representative family* of  $\mathcal{S}$ . The following result holds true due to Fomin et al. [17, 20].

► **Proposition 9.** *Let  $M = (U, \mathcal{I})$  be a linear matroid of rank  $n$  and  $p, q \leq n$  over a field  $\mathbb{F}$  and let  $\mathcal{S} = \{S_1, \dots, S_t\} \subseteq \mathcal{I}$  each having cardinality  $p$ . Then, there exists an algorithm that computes a *q-representative subfamily*  $\hat{\mathcal{S}} \subseteq_{\text{rep}}^q \mathcal{S}$  consisting of at most  $\binom{p+q}{q}$  sets using  $\mathcal{O}\left(\binom{p+q}{p}^2 tp^3 n^2 + t \binom{p+q}{q}^\omega np\right) + (n + |U|)^{\mathcal{O}(1)}$  field operations over  $\mathbb{F}$ . Here  $\omega < 2.37$  is the matrix multiplication exponent.*

Let  $G = (V, E)$  be an undirected graph and  $D_G = (V, A_E)$  is defined as follows. For every  $uv \in E(G)$ , we add  $(u, v)$  and  $(v, u)$  into  $A_E$  and fix  $v_r \in V$ . Since the definition of  $D_G$  is based on  $G = (V, E)$ , we call the pair  $(D_G, v_r)$  an *equivalent digraph of  $G$  with root  $v_r$* . Then, an *out-partition matroid with root  $v_r$*  for  $D_G$  is the partition matroid with ground set  $A_E$  where arcs are partitioned according to their heads and arcs  $(u, v_r)$  are dependent. Equivalently, what it means is that a set of arcs  $F \subseteq A_E$  is an *independent* in the out-partition matroid with root  $v_r$  if and only if  $v_r$  has in-degree 0 in  $F$  and every other vertex has in-degree at most 1 in  $F$ . The graphic matroid in the ground set  $A_E$  is the graphic matroid for  $G$  where every arc is represented by its underlying undirected edge and the antiparallel arcs  $(u, v), (v, u)$  represent distinct copies of  $uv$ . Then,  $\{(u, v), (v, u)\}$  becomes a dependent set. The following two propositions are proved by Agrawal et al. [2] and Einarson et al. [13] respectively.

► **Proposition 10** (Agrawal et al. [2]). *Let  $G = (V, E)$  be an undirected graph,  $v_r \in V$  and  $D_G = (V, A_E)$  as defined above. Then,  $G$  is  $p$ -edge-connected if and only if  $D_G$  has pairwise arc-disjoint out-branchings rooted at  $v_r$ .*

► **Proposition 11** (Einarson et al. [13]). *Let  $G = (V, E)$  be an undirected graph,  $v_r \in V$  and  $D_G = (V, A_E)$  as defined above. Then,  $F$  is the arc set of an out-branching rooted at  $v_r$  if and only if  $|F| = |V(G)| - 1$  and  $F$  is independent both in the out-partition matroid for  $D_G$  with root  $v_r$  and the graphic matroid for  $G$  with ground set  $A_E$ .*

Let  $G$  be an undirected graph,  $X \subseteq V(G)$  and  $p$  be an integer. It is not immediate whether in polynomial-time we can check whether there exists a *feasible* solution, that is, a set  $S \supseteq X$  such that  $G[S]$  is  $p$ -edge-connected. The following lemma illustrates that the above can be achieved in polynomial-time. In fact, in this case the input graph does not have to be a bounded degenerate graph.

► **Lemma 12** ( $\star$ ).<sup>2</sup> Let  $G = (V, E)$  be a connected undirected graph and  $X \subseteq V(G)$ . Then, we can check in polynomial-time if there exists a set  $S \supseteq X$  such that  $G[S]$  is a  $p$ -edge-connected subgraph.

We refer to Oxley [24] for more details on matroid theory and a survey by Panolan and Saurabh [25] for more information on use of matroids in FPT algorithms.

### 3 Algorithm for STEINER SUBGRAPH EXTENSION

This section is devoted to the proof of the main contribution of our paper. We first provide a singly exponential algorithm for STEINER SUBGRAPH EXTENSION (we restate below) when the input graph has bounded degeneracy. We assume that a fixed constant  $\eta$  is the degeneracy of  $G$ . We restate the problem definition.

STEINER SUBGRAPH EXTENSION

**Input:** An undirected graph  $G = (V, E)$ ,  $X \subseteq V(G)$  and integers  $k, p \in \mathbb{N}$ .

**Parameter:**  $k + p$

**Goal:** Is there  $S \supseteq X$  of size at most  $k$  such that  $G[S]$  is  $p$ -edge-connected?

Let  $(G, X, k)$  be given as an input instance and  $\sigma$  be a degeneracy sequence for the vertices of  $G - X$  witnessing that  $G - X$  is also an  $\eta$ -degenerate graph. Note that one can compute a degeneracy sequence easily in polynomial time by iteratively picking the minimum degree vertex, hence, we assume an ordering  $\sigma = (u_1, \dots, u_n)$  of the vertices of  $G - X$  is given along with the input. Due to Lemma 12, we can check if there exists a feasible solution  $S \supseteq X$  (not necessarily of size at most  $k$ ) such that  $G[S]$  is  $p$ -edge-connected subgraph. So, we can assume without loss of generality that a feasible solution actually exists. We first state a proposition that we use later in the proof of our result.

► **Proposition 13** ([13]). Let  $G = (V, E)$  be an undirected graph,  $v_r \in V$ , and  $D_G = (V, A_E)$  such that  $(D_G, v_r)$  is an equivalent digraph with root  $v_r$ . We also assume that  $M$  is a direct sum  $M_1 \oplus \dots \oplus M_{2p+1}$  as follows. The matroids  $M_1, M_3, \dots, M_{2p-1}$  are the copies of graphic matroid of  $G$  on ground set  $E$ , the matroids  $M_2, M_4, \dots, M_{2p}$  are the copies of out-partition matroids with ground set  $A_E$  with root  $v_r$ , and the matroid  $M_{2p+1}$  is a uniform matroid over  $A_E$  with rank  $p(k-1)$ . Furthermore, suppose that  $F \subseteq A_E$ , then the followings two are equivalent.

- (i)  $F$  is the set of  $p$  pairwise arc-disjoint out-branchings rooted at  $v_r$  in  $D_G[S]$  for some  $S \in \binom{V(G)}{k}$  and  $v_r \in S$ .
- (ii)  $|V(F)| = k$ ,  $|F| = p(k-1)$ ,  $v_r \in V(F)$ , and there is an independent set  $I$  in  $M$  such that every arc  $a \in F$  occurs in  $I$  precisely in its copies in matroids  $M_{2i-1}, M_{2i}$  and  $M_{2p+1}$  for some  $i \in \{1, \dots, p\}$ .

In addition, a linear representation of  $M$  and the truncation of  $M$  to  $\hat{M}$  of rank  $3p(k-1)$  can be computed in deterministic polynomial-time.

Our algorithm for STEINER SUBGRAPH EXTENSION works as follows. There are two cases. If  $X = \emptyset$ , then we choose an arbitrary vertex  $u \in V(G)$  and set  $X = \{u\}$ . There are  $n = |V(G)|$  possible choices of  $X$ . For each such choice we assign  $v_r = u \in X$ . Otherwise, it is already the case that  $X \neq \emptyset$ . Therefore, we can assume without loss of generality that  $X \neq \emptyset$ .

<sup>2</sup> Due to lack of space, the lemmas marked  $\star$  and the other omitted proofs can be found in the full version.

On the other hand, if  $X \neq \emptyset$  and  $G[X]$  is  $p$ -edge-connected, then we can trivially output yes-instance since  $|X| \leq k$ . So, we are in the situation that  $G[X]$  is not  $p$ -edge-connected and  $|X| < k$ . In the algorithm, we use the above characterization and representative sets framework to check if  $X$  can be extended to a  $p$ -edge-connected subgraph  $G[S]$  with at most  $k$  vertices. We fix an arbitrary vertex  $v_r \in X$ . Due to Proposition 13, there is an independent set  $I$  such that **(i)**  $X \subset V(I)$  and  $|V(I)| = k$ , **(ii)**  $|I| = 3p(k-1)$ , and **(iii)** every arc that is in  $I$  is represented precisely in three matroids  $M_{2i-1}, M_{2i}$  and  $M_{2p+1}$  for some  $i \in [p]$ .

We will build the set  $I$  via dynamic programming procedure. Since  $X$  is already included in  $V(I)$  it allows us to replace the first condition with  $|V(I) \setminus X| = k - |X|$ . The purpose of this dynamic programming is to construct a table that keeps track of  $|V(I) \setminus X|$  and  $|I|$ . Let  $\{v_1, \dots, v_{n'}\}$  be a degeneracy sequence of the vertices of  $G - X$  and  $\mathcal{X} = \{A_i \mid A_i = N(v_i) \cap \{v_1, \dots, v_{i-1}\} \setminus X\}$ . Each entry of the dynamic programming table  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  will contain a collection of independent sets of  $M$  that is a  $(3p(k-1) - q)$ -representative family of all the independent sets  $I$  of  $M$  such that  $|V(I) \setminus X| = i$ ,  $|I| = q$ ,  $Y = A_j \cap V(I)$ , the largest index of  $V(G) - X$  that occurs in  $V(I)$  is  $j$  and  $Z = A_\ell \cap V(I)$  for some  $\ell > j$ . Informally, it means that every independent set  $I$  has size  $q$  in  $M$ ,  $V(I)$  intersects  $A_j$  exactly in  $Y$ , and  $V(I)$  spans  $i$  vertices from  $G - X$ ,  $v_j \in V(I)$ , and  $V(I)$  has no vertex from  $\{v_{j+1}, \dots, v_{n'}\}$ . Furthermore, for every  $1 \leq j < n'$ ,  $V(I)$  intersects  $A_\ell$  exactly in  $Z$  for some  $\ell > j$ . Observe that for  $j = n'$ , there is no index  $\ell > j$ . Then we denote  $\ell = n' + 1$  and  $Z = \emptyset$  to keep the DP-states well-defined. We prove the following lemma that illustrates how a dynamic programming algorithm can construct all the entries of a table  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  for  $i \leq k - |X|, j \leq n', q \leq 3p(k-1), Y \subseteq A_j, Z \subseteq A_\ell$  and  $\ell > j$ . Indeed, if  $j = n'$ , then  $\ell = n' + 1$  and  $Z = \emptyset$ . Observe that there are at most  $2^n n$  possible choices of  $Y$  and  $2^n n$  possible choices for  $(Z, \ell)$  in the DP table  $\mathcal{T}$ . The following lemma illustrates how we compute the DP-table entries.

► **Lemma 14.** *Given matroid  $M$  of rank  $r = 3p(k-1)$  as described above, the entries of the table  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  for  $i \leq k - |X|, j \leq n', j < \ell, q \leq 3p(k-1), Y \subseteq A_j$  and  $Z \subseteq A_\ell$  can be computed in  $2^{\mathcal{O}(pk+\eta)} n^{\mathcal{O}(1)}$ -time.*

**Proof.** We describe a procedure  $\text{Construct}(\mathcal{T}[(i, j, q, Y), (Z, \ell)])$  for  $i \leq k - |X|, j \leq n'$  and  $q \leq 3p(k-1)$  as follows. Observe that every arc of  $I$  occurs in three copies, one in  $M_{2i-1}$ , one in  $M_{2i}$  and the other in  $M_{2p+1}$ . Given an arc  $a \in A_E$ , we use  $F_{a,i}$  to denote the set that contains the copies of  $a$  in  $M_{2i-1}, M_{2i}$  and  $M_{2p+1}$ . In the first part, we describe constructing the table entries  $\mathcal{T}[(0, 0, q, Y), (Z, \ell)]$  as follows.

- **(i)** For all  $1 \leq \ell \leq n'$ , we initialize  $\mathcal{T}[(0, 0, 0, \emptyset), (\emptyset, \ell)] = \{\emptyset\}$ .
- **(ii)** Consider the set of all the arcs in  $D_G[X]$ . We construct  $\mathcal{T}[(0, 0, q+3, \emptyset), (\emptyset, \ell)]$  from  $\mathcal{T}[(0, 0, q, \emptyset), (\emptyset, \ell)]$  as follows. For every  $I \in \mathcal{T}[(0, 0, q, \emptyset), (\emptyset, \ell)]$ , for every arc  $a \in D_G[X]$ , ( $1 \leq j \leq m$ ), and  $i \in \{1, \dots, p\}$ , we add  $I \cup F_{a,i}$  such that  $F_{a,i}$  extends  $I$ .
- **(iii)** Finally, we invoke Proposition 9 to reduce  $\mathcal{T}[(0, 0, q+3, \emptyset), (\emptyset, \ell)]$  into a  $(3p(k-1) - q - 3)$ -representative family of size  $2^{\mathcal{O}(pk+\eta)} n^{\mathcal{O}(1)}$ .

When we consider the table entries  $\mathcal{T}[(i, j, q+3, Y), (Z, \ell)]$  such that  $j = 0$ , observe that  $Y = Z = \emptyset$ . The reason is that for any  $I \in \mathcal{T}[(i, j, q+3, Y), (Z, \ell)]$  with the assumption of  $j = 0$  implies that  $i = 0$  and no vertex from  $G - X$  is part of  $V(I)$ . Since  $Y, Z \subseteq V(G) \setminus X$ , it must be that  $Y = Z = \emptyset$ . So, we consider only those entries in this first phase.

We analyze the running-time of the above process. Given  $\mathcal{T}[(0, 0, q, \emptyset), (\emptyset, \ell)]$ , computing  $\mathcal{T}[(0, 0, q+3, \emptyset), (\emptyset, \ell)]$  requires polynomial in the size of  $|\mathcal{T}[(0, 0, q, \emptyset), (\emptyset, \ell)]|$ . Then, computing a  $(3p(k-1) - q - 3)$ -representative family requires  $2^{\mathcal{O}(pk)}$ -time.

The process of computing table entries for  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  for  $Y \subseteq A_j$  and  $Z \subseteq A_\ell$  for  $\ell > j$  is more complex and needs more careful approach. We consider a lexicographic ordering of the indices  $((i, j, q, Y), (Z, \ell))$  and consider one by one as follows. For every

$i \geq 1$ , we compute the collection of independent sets in  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  using the collection of independent sets of  $\mathcal{T}[(i', j', q', Y'), (Z', \ell')]$  for  $j' < j$ ,  $j' < \ell'$ ,  $i = i' + 1$ , and  $q' < q$  as follows. We want to include  $v_j$  into  $V(I)$  such that  $|V(I) \setminus X| = i$ . We look at all the previous subproblems in the lexicographic ordering one by one. Suppose that  $I' \in \mathcal{T}[(i', j', q', Y'), (Z', \ell')]$ . By definition, it holds true that  $Y' = V(I') \cap A_{j'}$  and  $Z' = A_{\ell'} \cap V(I')$ . We say that  $I'$  is *extendable* to a set  $I \in \mathcal{T}[(i, j, q, Y), (Z, \ell)]$  if  $i = i' + 1$ ,  $Z' = Y$ , and  $\ell' = j$ . Observe that by definition  $V(I)$  and  $V(I')$  both intersect  $A_j$  exactly in  $Y$ , i.e.  $V(I) \cap A_j = Y$  and  $V(I') \cap A_j = Y$ . Moreover,  $j$  is the next larger index vertex included in  $V(I)$  after  $v_{j'}$ .

We construct the table entries of  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  using the table entries of  $\mathcal{T}[(i', j', q', Y'), (Z', \ell')]$  as follows.

Let  $d$  be the number of arcs that are incident to  $v_j$  such that either the other endpoints have lower indices or the other endpoint is in  $X$ . Formally, we consider the arcs that are either of the form  $(v_{j'}, v_j), (v_j, v_{j'}) \in A_E$  such that  $j' < j$  or of the form  $(u, v_j), (v_j, u)$  for some  $u \in X$ . Since  $G$  is an  $\eta$ -degenerate graph, observe that  $d \leq 2\eta + 2|X| \leq 2(k + \eta)$ . We create a set  $F$  of arcs as follows. For every arc  $a$  incident to  $v_j$ , we consider the sets  $F_{a,h}$  for every  $h \in [p]$ . We either add  $F_{a,h}$  into  $F$  for some  $h \in [p]$ , or do not add  $F_{a,h}$  into  $F$ . This ensures that there are  $(p+1)^d \leq (p+1)^{2\eta+2k}$  possible collections of arc-sets. For every nonempty such arc-set  $F$  and for every  $I' \in \mathcal{T}[(i', j', q', Y'), (Z', \ell)]$  satisfying  $Y = Z'$ ,  $i = i' + 1$  and  $\ell' = j$  (in other words  $I'$  that is extendable to a set in  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$ ), we add  $I' \cup F$  into  $\mathcal{T}[(i, j, q + |F|, Y), (Z, \ell)]$  when  $F$  extends  $I'$  and  $q = q' + |F|$ . Finally, invoke Proposition 9 to reduce  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  into its  $(3p(k-1) - q)$ -representative family containing at most  $2^{3p(k-1)}$  sets.

Observe that for every set  $I' \in \mathcal{T}[(i', j', q', Y'), (Z', \ell')]$ , there are at most  $(p+1)^{2\eta}$  sets  $I = I' \cup F$  that are added to the slot  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$ . Hence we have that

$$|\mathcal{T}[(i, j, q, Y), (Z, \ell)]| \leq (p+1)^{2|X|+2\eta} 2^{3pk} (p+k+\eta)^{\mathcal{O}(1)}$$

Since the number of indices is  $4^\eta n^2$ , the above implies that computing all the table entries can be performed in  $2^{\mathcal{O}(pk+\eta)} n^{\mathcal{O}(1)}$ -time. This completes the proof of the lemma.  $\blacktriangleleft$

Our next lemma ensures that the vertices of any independent set of size  $3p(k-1)$  computed by the above lemma induces a  $p$ -edge-connected subgraph.

**► Lemma 15.** *There is a set  $I \in \mathcal{T}[(k - |X|, j, 3p(k-1), Y), (Z, \ell)]$  for some  $Y \subseteq A_j$  and  $Z \subseteq A_\ell$  if and only if there exists  $S \supset X$  such that  $G[S]$  is a  $p$ -edge-connected subgraph of  $G$  with  $k$  vertices.*

**Proof (Sketch).**<sup>3</sup> First part of the proof is forward direction ( $\Rightarrow$ ). Let  $I \in \mathcal{T}[(k - |X|, j, 3p(k-1), Y), (Z, \ell)]$  be an independent set for some  $j \in [n']$ ,  $Y = A_j \cap V(I)$  and  $Z = V(I) \cap A_\ell$ . Let  $S = V(I)$  and it follows that  $|S \setminus X| = k - |X|$ . Note that  $|I| = 3p(k-1)$ . Due to Proposition 13, every arc of  $D_G$  occurs precisely in three copies. Recall that  $I$  was constructed by adding arcs  $F_{a,i}$  for some  $i \in [p]$  when the arc  $a$  incident to  $v \in V(G) \setminus X$  was added. But,  $F_{a,i}$  contains a copy of arc  $a$  in  $M_{2p+1}$ . Therefore, the construction of  $I$  ensures that no two distinct sets  $F_{a,i}$  and  $F_{a,i'}$  both are added. Hence  $D_G[V(I)]$  has  $p$  pairwise arc-disjoint out-branchings rooted at  $v_r$  and it follows from Proposition 10 that  $G[S]$  is  $p$ -edge-connected.

The other part of the proof is the backward direction ( $\Leftarrow$ ). Let  $G[S]$  be  $p$ -edge-connected with  $|S| = k$ , such that  $X \subseteq S$  and  $v_j$  is the vertex with highest index from  $\{v_1, \dots, v_n\} = V(G) \setminus X$ . It follows from Proposition 10 that there are  $p$  pairwise arc-disjoint out-branchings

<sup>3</sup> A complete detailed proof can be found in the full version.

## 45:10 Finding a Highly Connected Steiner Subgraph and its Applications

in  $D_G[S]$  rooted at  $v_r$ . Then, by Proposition 13, there is an independent set  $I$  such that  $V(I) = S$  containing the the set of arcs  $F \subseteq A_E$  such that  $|F| = p(k-1)$  and  $|I| = 3p(k-1)$ . We complete our proof by justifying that  $I$  is a candidate independent set of  $M$  for the slot  $\mathcal{T}[(k-|X|, j, 3p(k-1), Y), (Z, \ell)]$  for some  $Z = V(I) \cap A_\ell$  and  $\ell > j$ . If we prove that there is  $Y \subseteq N(v_j) \cap \{v_1, \dots, v_{j-1}\}$  such that  $\mathcal{T}[(k-|X|, j, 3p(k-1), Y), (Z, \ell)] \neq \emptyset$ , then we are done. We can give a proof of this claim by induction on  $i, j, q$  that  $\mathcal{T}[(k-|X|, j, 3p(k-1), Y), (Z, \ell)] \neq \emptyset$ . ◀

Using Lemma 14 and Lemma 15, we are ready to prove our theorem statement of our main result, i.e. Theorem 1 (we restate below).

► **Theorem 1.** STEINER SUBGRAPH EXTENSION can be solved in time  $2^{\mathcal{O}(pk+\eta)} n^{\mathcal{O}(1)}$ , where  $\eta$  is the degeneracy of the input graph.

**Proof.** We assume without loss of generality that  $X \neq \emptyset$  and  $G[X]$  is not  $p$ -edge-connected. Also, we use  $(G, X, k, p)$  to denote the input instance. Let  $\{v_1, \dots, v_{n'}\}$  be a degeneracy sequence of the vertices of  $G - X$  and consider an arbitrary vertex  $v_r \in X$ . The first step is to invoke Proposition 13 and construct a matroid  $M$ . We can also assume without loss of generality that  $(G, X, k-1, p)$  is a no-instance. Our next step is to invoke Lemma 14 and compute the table entries  $\mathcal{T}[(i, j, q, Y), (Z, \ell)]$  for all  $i \in \{1, \dots, k-|X|\}$ ,  $j \in [n']$ ,  $q \leq 3p(k-1)$ ,  $Y \subseteq A_j$ ,  $Z \subseteq A_\ell$  and  $\ell > j$ . It follows from Lemma 14 that all the table entries can be computed in  $2^{\mathcal{O}(pk+\eta)} n^{\mathcal{O}(1)}$ -time. Moreover, it follows from the Lemma 15 that for any  $I \in \mathcal{T}[(k-|X|, j, 3p(k-1), Y), (Z, \ell)]$ , it holds that  $G[V(I)]$  is a  $p$ -edge-connected subgraph. This completes the correctness proof of our algorithm. We finally output  $S = V(I)$  as the solution to the input instance. ◀

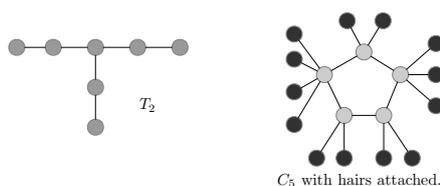
### 4 Applications of STEINER SUBGRAPH EXTENSION to some Graph Theoretic Problems

In this section, we describe some applications of our main result (Theorem 1) in parameterized algorithms. But before that, we prove the following lemma (proof is similar to the proof of Lemma 12).

► **Lemma 16** ( $\star$ ). *Let  $G$  be an input graph,  $p$  be a fixed constant, and  $\mathcal{G}$  be a polynomial-time recognizable hereditary graph class. Then, there exists a polynomial-time algorithm that can check if there is a set  $S \subseteq V(G)$  such that  $G - S \in \mathcal{G}$  and  $G[S]$  is  $p$ -edge-connected.*

The above lemma implies that for any polynomial-time recognizable (hereditary) graph class  $\mathcal{G}$ , we can test in polynomial-time if a feasible  $p$ -edge-connected vertex subset exists whose deletion results in a graph of class  $\mathcal{G}$ . Note that the class of all pathwidth-one graphs is a polynomial-time recognizable graph class. Moreover, for every fixed constant  $\eta$ , the class of all degree- $\eta$ -graphs, or the class of all  $\eta$ -treedepth graphs, or the class of all graphs with no  $P_\eta$  as subgraphs are all polynomial-time recognizable graph classes. So, we can test the existence of feasible solutions for all our problems in polynomial-time.

**Singly Exponential Algorithm for  $p$ -EDGE-CONNECTED- $\eta$ -DEGREE DELETION SET.** Now, we explain how Theorem 1 implies a singly exponential algorithm for  $p$ -EDGE-CONNECTED  $\eta$ -DEGREE DELETION SET problem. We formally state the problem below.



■ **Figure 1** An illustration of  $T_2$  and a cycle with hairs attached.

$p$ -EDGE-CONNECTED  $\eta$ -DEGREE DELETION SET ( $p$ -EDGE-CON-BDDS)

**Input:** An undirected graph  $G = (V, E)$  and an integer  $k$ .

**Parameter:**  $k$

**Goal:** Is there  $S \subseteq V(G)$  such that  $\Delta(G - S) \leq \eta$  and  $G[S]$  is  $p$ -edge-connected?

The following is the first application of our main result (Theorem 1). As we have a guarantee from Lemma 16 that we can test if an input graph has a feasible solution to our problems, we assume without loss of generality that the input graph actually has a feasible solution. We use this assumption for all our subsequent problems.

► **Corollary 2.**  $p$ -EDGE-CON-BDDS admits a  $2^{\mathcal{O}(pk+k\eta)}n^{\mathcal{O}(1)}$ -time algorithm.

The first part of the algorithm<sup>4</sup> for the above result uses enumeration of all minimal vertex subsets the removal of which results in a graph of maximum degree at most  $\eta$  and then it invokes Theorem 1.

**Singly Exponential Algorithm for  $p$ -EDGE-CON-PW1DS.** Now, we describe a singly exponential time algorithm for  $p$ -EDGE CONNECTED PATHWIDTH-1 VERTEX DELETION problem using Theorem 1. We formally define the problem as follows.

$p$ -EDGE-CON-PW1DS

**Input:** An undirected graph  $G = (V, E)$  and an integer  $k$ .

**Parameter:**  $k$

**Goal:** Is there  $S \subseteq V(G)$  such that  $G[S]$  is  $p$ -edge-connected and  $\text{pw}(G - S) \leq 1$ ?

We use the following characterization that are related to pathwidth one graphs.

► **Proposition 17** ([16, 4]). A graph  $G$  has pathwidth at most one if and only if it does not contain a cycle or  $T_2$  as a subgraph.

► **Proposition 18** ([26]). If  $G$  is a graph that does not contain any  $T_2, C_3, C_4$  as subgraphs, then each connected component of  $G$  is either a tree, or a cycle with zero or more pendant vertices (“hairs”) attached to it. (See Figure 1 for an illustration)

Observe that the graphs of pathwidth one do not have  $T_2, C_3, C_4$  as subgraphs. We prove the following lemma now.

► **Lemma 19** (\*). Let  $G$  be an undirected graph that does not have any  $T_2, C_3, C_4$  as subgraphs. Then, a 2-degeneracy sequence of  $G$  can be constructed in polynomial-time. Moreover, for every connected component  $D$  of  $G$ , a partition  $D = C \uplus P$  can be computed in polynomial-time such that  $C$  is an induced path (or cycle) and  $P$  is the set of pendant vertices attached to  $C$ .

<sup>4</sup> We refer to the full version for the proof

## 45:12 Finding a Highly Connected Steiner Subgraph and its Applications

It follows from Proposition 17 and Proposition 18 that any pathwidth one vertex deletion set must intersect all the subgraphs  $T_2, C_3, C_4$  of a graph. But, once we have a set  $X$  such that  $G - X$  has no  $T_2, C_3, C_4$  as subgraphs, then there are some connected components of  $G - X$  that can have cycles. In particular, due to Proposition 18, it holds that if a connected component of  $G - X$  has a cycle, then it must be a cycle with some (possibly empty set of) pendant hairs attached to it. Then, we would need to find  $S \supseteq X$  such that  $G[S]$  is  $p$ -edge-connected and  $S$  contains at least one vertex from each of these cycles. This requires us to design an algorithm that uses the ideas similar to Lemma 14 and Lemma 15 but also has to satisfy an additional condition. We state the following lemma and give a proof for completeness.

► **Lemma 20** ( $\star$ ). *Let  $G = (V, E)$  be an undirected graph and  $X \subseteq V(G)$  such that  $G - X$  has no  $T_2, C_3, C_4$  as subgraphs. Then, there exists an algorithm that runs in  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time and computes  $S \supseteq X$  of size at most  $k$  such that  $G - S$  has pathwidth at most one and  $G[S]$  is a  $p$ -edge-connected subgraph.*

Using the above lemma, we provide an  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time algorithm for  $p$ -EDGE CONNECTED PATHWIDTH-1 VERTEX DELETION problem as follows.

► **Theorem 3.**  *$p$ -EDGE-CON-PW1DS admits an algorithm that runs in  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time.*

**Proof.** Let  $(G, k)$  be an instance of  $p$ -EDGE CONNECTED PATHWIDTH-1 VERTEX DELETION problem. First we enumerate all minimal vertex subsets  $X$  of size at most  $k$  such that  $G - X$  has no  $T_2, C_3, C_4$  as subgraphs. Since  $T_2$  has 7 vertices,  $C_3$  has 3 vertices and  $C_4$  has 4 vertices, it takes  $\mathcal{O}^*(7^k)$ -time to enumerate all such subsets the deletion of which results in a graph that has no  $C_3, C_4, T_2$  as subgraphs. Let  $X$  be one such set such that  $G - X$  has no  $C_3, C_4, T_2$  as subgraphs. Due to Propositions 17 and 18, if  $D$  is a connected component of  $G - X$ , then either  $D$  is a caterpillar, or a cycle with hairs attached to it. It follows from Lemma 19 that there is a polynomial-time algorithm that gives a 2-degeneracy sequence  $\rho$  of the vertices of  $G - X$ . Moreover, if  $D$  is a connected component of  $G - X$ , then  $\rho$  provides a partition of  $D = C \uplus P$  such that  $C$  is a cycle and  $P$  is the set of hairs attached to  $C$ . In particular, the vertices of  $C$  are put first, followed by the vertices of  $P$  in  $\rho$ . Furthermore, putting the vertices of  $C$  first followed by the vertices of  $P$  gives a 2-degeneracy ordering of  $D$ . For each such subset  $X$ , we invoke Lemma 20 to give an algorithm that runs in  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time and outputs  $S$  such that  $X \subseteq S$  and  $G - S$  has pathwidth at most one. The correctness of this algorithm also follows from the proof of Lemma 20. This completes the proof of this theorem. ◀

### Singly Exponential Time Algorithms for $p$ -EDGE-CON- $\eta$ -TDDS and $p$ -EDGE-CON- $\eta$ -PVC.

Finally, we describe how we can get  $2^{\mathcal{O}(pk)}n^{\mathcal{O}(1)}$ -time algorithm for  $p$ -EDGE-CONNECTED  $\eta$ -TREEDEPTH DELETION SET and  $p$ -EDGE-CONNECTED  $\eta$ -PATH VERTEX COVER problems. Note that both  $p$  and  $\eta$  are fixed constants. We restate the problem definitions below.

$p$ -EDGE-CONNECTED  $\eta$ -TREEDEPTH DELETION SET ( $p$ -EDGE-CON- $\eta$ -TDDS)

**Input:** An undirected graph  $G = (V, E)$  and an integer  $k$ .

**Parameter:**  $k$

**Goal:** Is there  $S \subseteq V(G)$  such that  $G[S]$  is  $p$ -edge-connected and  $\text{td}(G - S) \leq \eta$ .

$p$ -EDGE-CONNECTED  $\eta$ -PATH VERTEX COVER

**Input:** An undirected graph  $G = (V, E)$  and an integer  $k$ .

**Parameter:**  $k$

**Goal:** Is there an  $S \subseteq V(G)$  with at most  $k$  vertices such that  $G - S$  has no  $P_\eta$  as subgraphs and  $G[S]$  is  $p$ -edge-connected?

It is clear from the problem definition that a set  $S$  is called an  $\eta$ -path vertex cover of  $G$  if  $G - S$  has no  $P_\eta$  as subgraph. The following proposition holds true for graphs of treedepth at most  $\eta$ .

► **Proposition 21** ([12]). *If a graph  $G$  has treedepth at least  $\eta + 1$ , then it has a connected subgraph  $H$  such that  $\text{td}(H) > \eta$  and  $|V(H)| \leq 2^{2^\eta}$ .*

The above proposition implies that  $\eta$ -TREEDPTH DELETION SET problem can be characterized as  $\mathcal{H}$ -HITTING SET problem where  $\mathcal{H}$  contains only subgraphs of bounded size. It follows from the definition that  $p$ -EDGE-CON- $\eta$ -PVC can be formulated in  $p$ -EDGE-CONNECTED  $\mathcal{H}$ -HITTING SET problem. It means that every minimal  $p$ -edge-connected  $\eta$ -treedepth deletion set contains a minimal  $\eta$ -treedepth deletion set and every minimal every minimal  $p$ -edge connected  $\eta$ -path vertex cover contains a minimal  $\eta$ -path vertex cover. We prove the following lemma that explains how we can construct a collection of all the minimal such solutions of size at most  $k$ .

► **Lemma 22** ( $\star$ ). *Given a (connected) undirected graph  $G = (V, E)$  and an integer  $k$ , the collection of all minimal  $\eta$ -treedepth deletion sets and the collection of all minimal  $\eta$ -path vertex covers can be obtained in  $2^{2^\eta k} n^{\mathcal{O}(1)}$ -time and  $\eta^k n^{\mathcal{O}(1)}$ -time respectively.*

The above lemma implies the next two results<sup>5</sup> as other applications of our main result.

► **Theorem 4.**  $p$ -EDGE-CON- $\eta$ -TDDS admits an algorithm that runs in  $2^{2^{2^\eta} + \mathcal{O}((p+\eta)k)} n^{2^{2^\eta}}$ -time.

Observe that a graph with no  $P_\eta$  as subgraph has treedepth at most  $\eta + 1$ . It means that such a graph also has bounded degeneracy. So, we have the following theorem.

► **Theorem 5.**  $p$ -EDGE-CON- $\eta$ -PVC admits an algorithm that runs in  $2^{\mathcal{O}((p+\eta)k)} n^{\mathcal{O}(1)}$ -time.

## 5 Conclusions and Future Work

There are several possible directions of future work. Our main result proves that STEINER SUBGRAPH EXTENSION is FPT when the removal of terminals results in a bounded degenerate graph. It is unclear if STEINER SUBGRAPH EXTENSION is FPT even when  $G - X$  is an arbitrary graph class and  $p$  is a fixed constant. Proving such a (positive or negative) result STEINER SUBGRAPH EXTENSION remains an interesting future work. If  $p = 2$ , then finding  $p$ -vertex/edge-connected steiner subgraph admits  $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ -time algorithm [19, 15]. It remains open if a 2-vertex-connected steiner subgraph can be obtained in  $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ -time even when the set of terminals is a vertex cover of the input graph. On the perspective of applications of Theorem 1, we have been successful in designing singly exponential-time FPT algorithms for  $p$ -EDGE-CON-BDDS,  $p$ -EDGE-CON- $\eta$ -TDDS,  $p$ -EDGE-CON-PW1DS and  $p$ -EDGE-CON- $\eta$ -PVC. But, our results do not capture several other graph classes. For instance,

<sup>5</sup> The proofs of the next two theorems can be found in the full version.

the above algorithm crucially relies that all minimal vertex deletion sets without connectivity requirements can be enumerated in  $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ -time and that a bounded degeneracy sequence can be computed in polynomial-time. Therefore, obtaining an FPT algorithm with singly exponential running time for each of FEEDBACK VERTEX SET, CLUSTER VERTEX DELETION, COGRAPH VERTEX DELETION with  $p$ -edge-connectivity constraints (even with  $p = 2$ ) also remain interesting open problems.

---

## References

- 1 Ankit Abhinav, Susobhan Bandopadhyay, Aritra Banik, and Saket Saurabh. Parameterized algorithms for finding highly connected solution. *Theor. Comput. Sci.*, 942:47–56, 2023.
- 2 Akanksha Agrawal, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Fast exact algorithms for survivable network design with uniform requirements. *Algorithmica*, 84(9):2622–2641, 2022.
- 3 MohammadHossein Bateni, Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. A PTAS for planar group steiner tree via spanner bootstrapping and prize collecting. 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 570–583. ACM, 2016.
- 4 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly( $k$ ) kernel. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2006.
- 5 Radovan Cervený and Ondrej Suchý. Faster FPT algorithm for 5-path vertex cover. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 32:1–32:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 6 Radovan Cervený and Ondrej Suchý. Generating faster algorithms for d-path vertex cover. *CoRR*, abs/2111.05896, 2021. [arXiv:2111.05896](https://arxiv.org/abs/2111.05896).
- 7 Rajesh Hemant Chitnis, Andreas Emil Feldmann, Mohammad Taghi Hajiaghayi, and Dániel Marx. Tight bounds for planar strongly connected steiner subgraph with fixed number of terminals (and extensions). *SIAM J. Comput.*, 49(2):318–364, 2020.
- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 Celina M. H. de Figueiredo, Raul Lopes, Alexsander Andrade de Melo, and Ana Silva. Parameterized algorithms for steiner tree and dominating set: Bounding the leafage by the vertex leafage. In Petra Mutzel, Md. Saidur Rahman, and Slamun, editors, *WALCOM: Algorithms and Computation – 16th International Conference and Workshops, WALCOM 2022, Jember, Indonesia, March 24-26, 2022, Proceedings*, volume 13174 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2022.
- 10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 12 Zdenek Dvorák, Archontia C. Giannopoulou, and Dimitrios M. Thilikos. Forbidden graphs for tree-depth. *Eur. J. Comb.*, 33(5):969–979, 2012.
- 13 Carl Einarson, Gregory Z. Gutin, Bart M. P. Jansen, Diptapriyo Majumdar, and Magnus Wahlström.  $p$ -edge/vertex-connected vertex cover: Parameterized and approximation algorithms. *J. Comput. Syst. Sci.*, 133:23–40, 2023.

- 14 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. 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 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
- 15 Andreas Emil Feldmann, Anish Mukherjee, and Erik Jan van Leeuwen. The parameterized complexity of the survivable network design problem. In Karl Bringmann and Timothy Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 37–56. SIAM, 2022.
- 16 Michael R. Fellows and Michael A. Langston. On search, decision and the efficiency of polynomial-time algorithms (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 501–512. ACM, 1989.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016.
- 18 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Trans. Algorithms*, 13(3):35:1–35:35, 2017.
- 19 Pinar Heggernes, Pim van ’t Hof, Dániel Marx, Neeldhara Misra, and Yngve Villanger. On the parameterized complexity of finding separators with non-hereditary properties. *Algorithmica*, 72(3):687–713, 2015.
- 20 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018.
- 21 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. On subexponential parameterized algorithms for steiner tree and directed subset TSP on planar graphs. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 474–484. IEEE Computer Society, 2018.
- 22 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- 23 Zeev Nutov. Parameterized algorithms for node connectivity augmentation problems. *CoRR*, abs/2209.06695, 2022. [arXiv:2209.06695](https://arxiv.org/abs/2209.06695).
- 24 James G. Oxley. *Matroid theory*. Oxford University Press, 1992.
- 25 Fahad Panolan and Saket Saurabh. Matroids in parameterized complexity and exact algorithms. In *Encyclopedia of Algorithms*, pages 1203–1206. Springer, 2016.
- 26 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.



# FPT Approximation and Subexponential Algorithms for Covering Few or Many Edges

Fedor V. Fomin  

University of Bergen, Norway

Petr A. Golovach  

University of Bergen, Norway

Tanmay Inamdar  

University of Bergen, Norway

Tomohiro Koana  

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

---

## Abstract

We study the  $\alpha$ -FIXED CARDINALITY GRAPH PARTITIONING ( $\alpha$ -FCGP) problem, the generic local graph partitioning problem introduced by Bonnet et al. [Algorithmica 2015]. In this problem, we are given a graph  $G$ , two numbers  $k, p$  and  $0 \leq \alpha \leq 1$ , the question is whether there is a set  $S \subseteq V$  of size  $k$  with a specified coverage function  $\text{cov}_\alpha(S)$  at least  $p$  (or at most  $p$  for the minimization version). The coverage function  $\text{cov}_\alpha(\cdot)$  counts edges with exactly one endpoint in  $S$  with weight  $\alpha$  and edges with both endpoints in  $S$  with weight  $1 - \alpha$ .  $\alpha$ -FCGP generalizes a number of fundamental graph problems such as DENSEST  $k$ -SUBGRAPH, MAX  $k$ -VERTEX COVER, and MAX  $(k, n - k)$ -CUT.

A natural question in the study of  $\alpha$ -FCGP is whether the algorithmic results known for its special cases, like MAX  $k$ -VERTEX COVER, could be extended to more general settings. One of the simple but powerful methods for obtaining parameterized approximation [Manurangsi, SOSA 2019] and subexponential algorithms [Fomin et al. IPL 2011] for MAX  $k$ -VERTEX COVER is based on the greedy vertex degree orderings. The main insight of our work is that the idea of greedy vertex degree ordering could be used to design fixed-parameter approximation schemes (FPT-AS) for  $\alpha > 0$  and the subexponential-time algorithms for the problem on apex-minor free graphs for maximization with  $\alpha > 1/3$  and minimization with  $\alpha < 1/3$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Fixed parameter tractability; Theory of computation  $\rightarrow$  Approximation algorithms analysis

**Keywords and phrases** Partial Vertex Cover, Approximation Algorithms, Max Cut

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.46

**Funding** *Fedor V. Fomin*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).

*Petr A. Golovach*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).

*Tanmay Inamdar*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (LOPRE grant no. 819416).

*Tomohiro Koana*: Supported by the DFG project DiPa (NI 369/21).

## 1 Introduction

In this work, we study a broad class of problems called  $\alpha$ -FIXED CARDINALITY GRAPH PARTITIONING ( $\alpha$ -FCGP), originally introduced by Bonnet et al. [2]<sup>1</sup>. The input is a graph  $G = (V, E)$ , two non-negative integers  $k, p$ , and a real number  $0 \leq \alpha \leq 1$ . The question is whether there is a set  $S \subseteq V$  of size exactly  $k$  with  $\text{cov}_\alpha(S) \geq p$  ( $\text{cov}_\alpha(S) \leq p$  for the minimization variant), where

---

<sup>1</sup> Bonnet et al. [2] called the problem “local graph partitioning problem”, however we adopt the nomenclature from Koana et al. [19].



$$\text{cov}_\alpha(S) := (1 - \alpha) \cdot m(S) + \alpha \cdot m(S, V \setminus S).$$

Here,  $m(S)$  is the number of edges with both endpoints in  $S$ , and  $m(S, V \setminus S)$  is the number of edges with one endpoint in  $S$  and other in  $V \setminus S$ . We will call the maximization and minimization problems MAX  $\alpha$ -FCGP and MIN  $\alpha$ -FCGP, respectively. This problem generalizes many problems, namely, DENSEST  $k$ -SUBGRAPH (for  $\alpha = 0$ ), MAX  $k$ -VERTEX COVER<sup>2</sup> (for  $\alpha = 1/2$ ), MAX  $(k, n - k)$ -CUT (for  $\alpha = 1$ ), and their minimization counterparts.

Although there are plethora of publications that study these special cases, the general  $\alpha$ -FCGP has not received much attention, except for the work of Bonnet et al. [2], Koana et al. [19], and Schachnai and Zehavi [23]. In this paper, we aim to demonstrate the wider potential of the existing algorithms designed for specific cases, such as MAX  $k$ -VERTEX COVER, by presenting an algorithm that can handle the more general problem of  $\alpha$ -FCGP. Algorithms for these specific cases often rely on greedy vertex degree orderings. For instance, Manurangsi [20], showing that a  $(1 - \varepsilon)$ -approximate solution can be found in the set of  $\mathcal{O}(k/\varepsilon)$  vertices with the largest degrees, gave a  $(1 - \varepsilon)$ -approximation algorithm for MAX  $k$ -VERTEX COVER that runs in time  $(1/\varepsilon)^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ . Fomin et al. [14] gave a  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ -time algorithm for MAX  $k$ -VERTEX COVER on apex-minor graphs via bidimensionality arguments, by showing that an optimal solution  $S$  is adjacent to every vertex of degree at least  $d + 1$ , where  $d$  is the minimum degree over vertices in  $S$ . In this work, we will give approximation algorithms as well as subexponential-time algorithms for apex-minor free graphs exploiting the greedy vertex ordering.

For approximation algorithms, we will show that both MAX  $\alpha$ -FCGP and MIN  $\alpha$ -FCGP admit *FPT Approximation Schemes* (FPT-AS) for  $\alpha > 0$ , i.e., there is an algorithm running in time  $f(k, \alpha, \varepsilon) \cdot n^{\mathcal{O}(1)}$  that finds a set  $S$  of size  $k$  with  $\text{cov}_\alpha(S) \geq (1 - \varepsilon) \cdot \text{OPT}$  (or  $\text{cov}_\alpha(S) \leq (1 + \varepsilon) \cdot \text{OPT}$  for the minimization variant), where  $\text{OPT}$  denotes the optimal value of  $p$ . Previously, the special cases were known to admit FPT approximation schemes; see [22, 16, 17, 20] for  $\alpha = 1/2$  and [2] for  $\alpha = 1$ . In particular, the state-of-the-art running time for  $\alpha = 1/2$  is the aforementioned algorithm of Manurangsi that runs in time  $(1/\varepsilon)^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$  for maximization (also for the minimization variant). For  $\alpha = 0$ , the situation is more negative; MAX  $\alpha$ -FCGP (namely, DENSEST  $k$ -SUBGRAPH) does not admit any  $o(k)$ -approximation algorithm with running time  $f(k) \cdot n^{\mathcal{O}(1)}$  under the Strongish Planted Clique Hypothesis [21]. MIN  $\alpha$ -FCGP is also hard to approximate when  $\alpha = 0$  since it encompasses INDEPENDENT SET as a special case for  $p = 0$ .

Next, we discuss the regime of subexponential-time algorithms. Amini et al. [1] showed that MAX  $k$ -VERTEX COVER is FPT on graphs of bounded degeneracy, including planar graphs, giving a  $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ -time algorithm. They left it open whether it can be solved in time  $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ . This was answered in the affirmative by Fomin et al. [14], who showed that MAX  $k$ -VERTEX COVER on apex-minor free graphs can be solved in time  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$  time. Generalizing this result, we give a  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ -time algorithm for MAX  $\alpha$ -FCGP with  $\alpha > 1/3$  and MIN  $\alpha$ -FCGP with  $\alpha < 1/3$ . The complexity landscape of MAX  $\alpha$ -FCGP with  $\alpha < 1/3$  (and MIN  $\alpha$ -FCGP with  $\alpha > 1/3$ ) is not well understood. It is a long-standing open question whether DENSEST  $k$ -SUBGRAPH on planar graphs is NP-hard [4]. Note that the special case CLIQUE is trivially polynomial-time solvable on planar graphs because a clique on 5 vertices does not admit a planar embedding.

---

<sup>2</sup> This problem is also referred to as PARTIAL VERTEX COVER.

### Further related work

As mentioned, special cases of  $\alpha$ -FCGP when  $\alpha \in \{0, 1/2, 1\}$  have been extensively studied. For instance, the W[1]-hardness for the parameter  $k$  has been long known for these special cases [3, 11, 15]. Both MAX  $\alpha$ -FCGP and MIN  $\alpha$ -FCGP are actually W[1]-hard for every  $\alpha \in [0, 1]$  with the exception  $\alpha \neq 1/3$ , as can be seen from a parameterized reduction from CLIQUE and INDEPENDENT SET on regular graphs. Note that  $\alpha$ -FIXED CARDINALITY GRAPH PARTITIONING becomes trivial when  $\alpha = 1/3$  because  $\text{cov}_\alpha(S) = \frac{1}{3} \cdot \sum_{v \in S} d(v)$  for any  $S \subseteq V$  where  $d(v)$  is the degree of  $v$ .

Bonnet et al. [2] gave a  $(\Delta k)^{2k} \cdot n^{\mathcal{O}(1)}$ -time algorithm for  $\alpha$ -FCGP where  $\Delta$  is the maximum degree. They also gave an algorithm with running time  $\Delta^k \cdot n^{\mathcal{O}(1)}$  for MAX  $\alpha$ -FCGP with  $\alpha > 1/3$  and MIN  $\alpha$ -FCGP with  $\alpha < 1/3$ . This result was strengthened by Schachnai and Zehavi [23]; they gave a  $4^{k+o(k)} \Delta^k \cdot n^{\mathcal{O}(1)}$ -time algorithm for any value of  $\alpha$ . Koana et al. [19] showed that MAX  $\alpha$ -FCGP admits polynomial kernels on sparse families of graphs when  $\alpha > 1/3$ . For instance, MAX  $\alpha$ -FCGP admits a  $k^{\mathcal{O}(d)}$ -sized kernel where  $d$  is the degeneracy of the input graph. They also showed analogous results for MIN  $\alpha$ -FCGP with  $\alpha < 1/3$ .

### Preliminaries

For an integer  $n$ , let  $[n]$  denote the set  $\{1, \dots, n\}$ .

We use the standard graph-theoretic notation and refer to the textbook of Diestel [10] for undefined notions. In this work, we assume that all graphs are simple and undirected. For a graph  $G$  and a vertex set  $S$ , let  $G[S]$  be the subgraph of  $G$  induced by  $S$ . For a vertex  $v$  in  $G$ , let  $d(v)$  be its *degree*, i.e., the number of its neighbors. For vertex sets  $X, Y$ , let  $m(X) := |\{uv \in E \mid u, v \in X\}|$  and  $m(X, Y) := |\{uv \in E \mid u \in X, v \in Y\}|$ . In this work, an optimal solution for MAX  $\alpha$ -FCGP (and MIN  $\alpha$ -FCGP) is a vertex set  $S$  of size  $k$  such that  $\text{cov}_\alpha(S) \geq \text{cov}_\alpha(S')$  (resp.,  $\text{cov}_\alpha(S) \leq \text{cov}_\alpha(S')$ ) for every vertex set of size  $k$ . A graph  $H$  is a *minor* of  $G$  if a graph isomorphic to  $H$  can be obtained from  $G$  by vertex and edge removals and edge contractions. Given a graph  $H$ , a family of graph  $\mathcal{H}$  is said to be  *$H$ -minor free* if there is no  $G \in \mathcal{H}$  having  $H$  as a minor. A graph  $H$  is an *apex graph* if  $H$  can be made planar by the removal of a single vertex.

We refer to the textbook of Cygan et al. [5] for an introduction to Parameterized Complexity and we refer to the paper of Marx [22] for an introduction to the area of parameterized approximation.

## 2 FPT Approximation Algorithms

In this section, we design an FPT Approximation Scheme for MAX  $\alpha$ -FCGP as well as MIN  $\alpha$ -FCGP parameterized by  $k$  and  $\alpha$ , assuming  $\alpha > 0$ . More specifically, we prove the following theorem.

► **Theorem 1.** *For any  $0 < \alpha \leq 1$  and  $0 < \epsilon \leq 1$ , MAX  $\alpha$ -FCGP and MIN  $\alpha$ -FCGP each admits an FPT-AS parameterized by  $k$ ,  $\epsilon$  and  $\alpha$ . More specifically, given a graph  $G = (V, E)$  and an integer  $k$ , there exists an algorithm that runs in time  $f(k, \alpha, \epsilon) \cdot n^{\mathcal{O}(1)}$ , and finds a set  $S \subseteq V$  such that  $\text{cov}_\alpha(S) \geq (1 - \epsilon) \cdot \text{cov}_\alpha(O)$  for MAX  $\alpha$ -FCGP and  $\text{cov}_\alpha(S) \leq (1 + \epsilon) \cdot \text{cov}_\alpha(O)$  for MIN  $\alpha$ -FCGP, where  $O \subseteq V$  is an optimal solution.*

For the case that  $\text{OPT} := \text{cov}_\alpha(O)$  is large, the following greedy argument will be helpful.

► **Lemma 2.** For MAX  $\alpha$ -FCGP, let  $S$  be the set of  $k$  vertices with the largest degrees. Then,  $\text{cov}_\alpha(S) \geq \text{OPT} - 2k^2$ . For MIN  $\alpha$ -FCGP, let  $S$  be the set of  $k$  vertices with the smallest degrees. Then,  $\text{cov}_\alpha(S) \leq \text{OPT} + 2k^2$ .

**Proof.** Without loss of generality, we assume that  $O \neq S$ . Let  $S \setminus O = \{y_1, y_2, \dots, y_t\}$ , and  $O \setminus S = \{w_1, w_2, \dots, w_t\}$ , where  $1 \leq t \leq k$ . Here, we index the vertices so that  $d(y_i) \geq d(y_j)$  and  $d(w_i) \geq d(w_j)$  (for MIN  $\alpha$ -FCGP,  $d(y_i) \leq d(y_j)$  and  $d(w_i) \leq d(w_j)$ ) for  $i < j$ . Note that due to the choice of  $S$ , it holds that  $d(y_i) \geq d(w_i)$  ( $d(y_i) \leq d(w_i)$  for MIN  $\alpha$ -FCGP) for each  $1 \leq i \leq t$ .

Now we define a sequence of solutions  $O_0, O_1, \dots, O_t$ , where  $O_0 = O$ , and for each  $1 \leq i \leq t$ ,  $O_i := (O_{i-1} \setminus \{w_i\}) \cup \{y_i\}$ . Note that  $O_t = S$ . We claim that for each  $1 \leq i \leq t$ ,  $\text{cov}_\alpha(O_i) \geq \text{cov}_\alpha(O_{i-1}) - 2k$  for MAX  $\alpha$ -FCGP and  $\text{cov}_\alpha(O_i) \leq \text{cov}_\alpha(O_{i-1}) + 2k$  for MIN  $\alpha$ -FCGP. To this end, we note that  $O_i$  is obtained from  $O_{i-1}$  by removing  $w_i$  and adding  $y_i$ . Thus,  $\text{cov}_\alpha(O_i) = \text{cov}_\alpha(O_{i-1}) - (\alpha m_1 + ((1 - \alpha) - \alpha) \cdot m_2) + \alpha m_3 + ((1 - \alpha) - \alpha) \cdot m_4$ , where

$$\begin{aligned} m_1 &:= m(\{w_i\}, V \setminus O_{i-1}), & m_2 &:= m(\{w_i\}, O_{i-1} \setminus \{w_i\}), \\ m_3 &:= m(\{y_i\}, V \setminus O_i), & m_4 &:= m(\{y_i\}, O_i \setminus \{w_i\}). \end{aligned}$$

Observe that  $d(w_i) - k \leq m_1 \leq d(w_i)$ ,  $d(y_i) - k \leq m_3 \leq d(y_i)$ , and  $0 \leq m_2, m_4 \leq k$ . We consider MAX  $\alpha$ -FCGP first. We have that

$$\begin{aligned} \text{cov}_\alpha(O_i) &= \text{cov}_\alpha(O_{i-1}) + \alpha(m_3 - m_1) + (1 - 2\alpha)(m_4 - m_2) \\ &\geq \text{cov}_\alpha(O_{i-1}) + \alpha(m_3 - m_1) - |(1 - 2\alpha)(m_4 - m_2)|. \end{aligned}$$

Since  $m_3 - m_1 \geq d(y_i) - d(w_i) - k \geq -k$  and  $|(1 - 2\alpha)(m_4 - m_2)| \leq k$ , we obtain  $\text{cov}_\alpha(O_i) \geq \text{cov}_\alpha(O_{i-1}) - 2k$ , regardless of the value of  $\alpha$ . We consider MIN  $\alpha$ -FCGP next. It holds that

$$\begin{aligned} \text{cov}_\alpha(O_i) &= \text{cov}_\alpha(O_{i-1}) + \alpha(m_3 - m_1) + (1 - 2\alpha)(m_4 - m_2) \\ &\leq \text{cov}_\alpha(O_{i-1}) + \alpha(m_3 - m_1) + |(1 - 2\alpha)(m_4 - m_2)|. \end{aligned}$$

Since  $m_3 - m_1 \leq d(y_i) - d(w_i) + k \leq k$  and  $|(1 - 2\alpha)(m_4 - m_2)| \leq k$ , we obtain  $\text{cov}_\alpha(O_i) \leq \text{cov}_\alpha(O_{i-1}) + 2k$ , regardless of the value of  $\alpha$ .

Therefore,  $\text{cov}_\alpha(O_t) \geq \text{cov}_\alpha(O_0) - 2kt \geq \text{OPT} - 2k^2$  for MAX  $\alpha$ -FCGP and  $\text{cov}_\alpha(O_t) \leq \text{cov}_\alpha(O_0) + 2kt \leq \text{OPT} + 2k^2$  for MIN  $\alpha$ -FCGP. ◀

Lemma 2 allows us to find an approximate solution when  $\text{OPT}$  is sufficiently large. The case that  $\text{OPT}$  is small remains. We use different approaches for MAX  $\alpha$ -FCGP and MIN  $\alpha$ -FCGP.

### Algorithm for MAX $\alpha$ -FCGP

Let  $v_1$  be a vertex with the largest degree. Our algorithm considers two cases depending on whether  $d(v_1) > \Delta := \frac{2k^2}{\epsilon\alpha} + k$ . If  $d(v_1) > \Delta$ , we can argue that the set  $S$  from Lemma 2 a  $(1 - \epsilon)$ -approximate solution. To that end, we make the following observation.

► **Observation 3.** If  $d(v_1) > \Delta$ , then  $2k^2 \leq \epsilon \cdot \text{cov}_\alpha(S)$ .

**Proof.** Note that  $m(S, V \setminus S) = \sum_{u \in S} m(\{u\}, V \setminus S) \geq m(\{v_1\}, V \setminus S) \geq d(v_1) - k$ , where the inequality follows from the fact that at most  $k$  edges incident to  $v_1$  can have the other endpoint in  $S$ . This implies that

$$\text{cov}_\alpha(S) \geq \alpha \cdot m(S, V \setminus S) \geq \alpha \cdot (d(v_1) - k) \geq \frac{2k^2}{\epsilon}.$$

Where we use the assumptions that  $0 < \alpha \leq 1$  and  $d(v_1) \geq \Delta$ . ◀

Thus, for  $d(v_1) > \Delta$ , we have  $\text{OPT} \leq \text{cov}_\alpha(S) + 2k^2 \leq (1 + \varepsilon) \cdot \text{cov}_\alpha(S)$ , and thus  $\text{cov}_\alpha(S) \geq (1 - \varepsilon) \cdot \text{OPT}$ .

So assume that  $d(v_1) < \Delta$ . In this case, the maximum degree of the graph is bounded by  $\Delta$ . Let  $O \subseteq V$  be an optimal solution. Then the total number of edges contributing to  $\text{cov}_\alpha(O)$  is bounded by  $k\Delta = \mathcal{O}(k^3/\alpha\epsilon)$ . Let  $Q$  be the set of vertices in  $V \setminus O$  that have a neighbor in  $O$ , and note that  $|Q| = \mathcal{O}(k^3/\alpha\epsilon)$ . Let  $z = |O| + |Q|$ , and note that  $z = \mathcal{O}(k^3/\alpha\epsilon)$ .

We first guess the structure of the subgraph  $G' = (O \cup Q, E')$ , where  $E'$  consists of all edges with at least one endpoint in  $O$ . For each guess for  $G'$ , we check whether there exists a subgraph in  $G$  that is isomorphic to  $G'$ . Over all guesses where we find an isomorphic subgraph, we return the solution maximizing the  $\text{cov}_\alpha(\cdot)$  value. Note that the number of guesses is bounded by  $2^{z^2} = g(k, \alpha, \epsilon)$ . Since the maximum degree of  $G$  is bounded by  $\Delta$ , and the number of vertices in the subgraph corresponding to each guess is  $z$ , we can solve each instance of SUBGRAPH ISOMORPHISM in time  $2^{\mathcal{O}(z\Delta)} z! \cdot n^{\mathcal{O}(1)} = g'(k, \alpha, \epsilon) \cdot n^{\mathcal{O}(1)}$  using random separation, e.g., Theorem 5.7 in [5]. Thus, overall, the running time of the algorithm is bounded by some  $f(k, \alpha, \epsilon) \cdot n^{\mathcal{O}(1)}$ . Combining both cases, we conclude the proof of Theorem 1.

### Algorithm for MIN $\alpha$ -FCGP

For MIN  $\alpha$ -FCGP, our algorithm considers two cases depending on the value of OPT. If  $\text{OPT} \geq \frac{2k^2}{\epsilon}$ , then our algorithm returns the set  $S$  from Lemma 2. Note that  $\text{cov}_\alpha(S) \leq \text{OPT} + 2k^2 \leq (1 + \varepsilon) \cdot \text{OPT}$ .

Now suppose that  $\text{OPT} < \frac{2k^2}{\epsilon}$ . In this case, we know that  $O$  cannot contain a vertex of degree larger than  $\Delta = \frac{2k^2}{\alpha\epsilon} + k$ , for otherwise,  $\text{cov}_\alpha(O) > \alpha(\Delta - k) \geq \text{OPT}$ , which is a contradiction.

In this case, we can guess the structure of  $G' = (O \cup N(O), E')$ , where  $E'$  consists of all edges with at least one endpoint in  $E'$ . Then, we can find a subgraph isomorphic to  $G'$  using an FPT algorithm (we can delete the edges between all vertices whose degree is larger than  $\Delta$ ). This takes FPT time.

Since the value of OPT is unknown to us, we cannot directly conclude which case is applicable. So we find a solution for each case and return a better one.

## 3 Subexponential FPT Algorithm for MAX $\alpha$ -FCGP on Apex-Minor Free Graphs

Fomin et al. [14] showed that PARTIAL VERTEX COVER on apex-minor free graphs can be solved in time  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ . In this section, we will prove its generalization to MAX  $\alpha$ -FCGP as well as MIN  $\alpha$ -FCGP:

- **Theorem 4.** *For an apex graph  $H$ , let  $\mathcal{H}$  be a family of  $H$ -minor free graphs.*
- *For any  $\alpha \geq 1/3$ , MAX  $\alpha$ -FCGP for  $\mathcal{H}$  can be solved in  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$  time.*
  - *For any  $\alpha \leq 1/3$ , MIN  $\alpha$ -FCGP for  $\mathcal{H}$  can be solved in  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$  time.*

We will give a proof for the maximization variant. The minimization variant follows analogously. Let  $\sigma = v_1, v_2, \dots, v_n$  be an ordering of vertices of  $V$  in the non-increasing order of degrees, with ties broken arbitrarily. That is,  $d(v_1) \geq d(v_2) \geq \dots \geq d(v_{n-1}) \geq d(v_n)$ . We will denote the graph by  $G = (V_\sigma, E)$  to emphasize the fact that the vertex set is ordered w.r.t.  $\sigma$ . We also let  $V_\sigma^j = \{v_1, \dots, v_j\}$ . We first prove the following lemma.

► **Lemma 5.** *Let  $G = (V_\sigma, E)$  be a yes-instance for MAX  $\alpha$ -FCGP, where  $1/3 \leq \alpha \leq 1$ . Let  $C = \{u_{i_1}, u_{i_2}, \dots, u_{i_k}\}$  be the lexicographically smallest solution for MAX  $\alpha$ -FCGP and  $u_{i_k} = v_j$  for some  $j$ . Then  $C$  is a dominating set of size  $k$  for  $G[V_\sigma^j]$ .*

**Proof.** Suppose for the contradiction that  $C$  is not a dominating set for  $G[V_\sigma^j]$ . Then, there exists a vertex  $v_i$  with  $1 \leq i < j$  such that  $N[v_i] \cap C = \emptyset$ . Set  $C' = (C \setminus \{v_j\}) \cup \{v_i\}$ . Note that  $d(v_i) \geq d(v_j)$ . Define the following:

$$\begin{aligned} m_1 &= m(\{v_j\}, V \setminus C), \\ m_2 &= m(\{v_j\}, C \setminus \{v_j\}), \\ m_3 &= m(\{v_i\}, (V \setminus C) \cup \{v_j\}) = d(v_i), \\ m_4 &= m(\{v_i\}, C \setminus \{v_j\}) = 0. \end{aligned}$$

We will show that  $C'$  is another solution for the MAX  $\alpha$ -FCGP instance. Since  $C' \setminus \{v_i\} = C \setminus \{v_j\}$ , it suffices to show that

$$\text{cov}_\alpha(C') - \text{cov}_\alpha(C) = (\text{cov}_\alpha(C') - \text{cov}_\alpha(C' \setminus \{v_i\})) - (\text{cov}_\alpha(C) - \text{cov}_\alpha(C \setminus \{v_j\}))$$

is nonnegative. By definition,

$$\begin{aligned} \text{cov}_\alpha(C') - \text{cov}_\alpha(C' \setminus \{v_i\}) &= \alpha \cdot m_3 + ((1 - \alpha) - \alpha) \cdot m_4 = \alpha \cdot d(v_i) \text{ and} \\ \text{cov}_\alpha(C) - \text{cov}_\alpha(C \setminus \{v_j\}) &= \alpha \cdot m_1 + ((1 - \alpha) - \alpha) \cdot m_2 \leq \alpha \cdot (m_1 + m_2) = \alpha \cdot d(v_j), \end{aligned} \tag{1}$$

where the inequality is due to the assumption that  $\alpha \geq 1/3$ . Therefore,

$$\text{cov}_\alpha(C') - \text{cov}_\alpha(C) = \alpha \cdot (d(v_i) - d(v_j)) \geq 0,$$

which is a contradiction to the assumption that  $C$  is the lexicographically smallest solution for MAX  $\alpha$ -FCGP. ◀

In view of Lemma 5, we can use the following approach to search for the lexicographically smallest solution  $C$ . First, we guess the last vertex  $v_j$  of  $C$  in the ordering  $\sigma$ , i.e., we search for a solution  $C$  such that  $v_j \in C$  and  $C \subseteq V_\sigma^j$ . If  $G[V_\sigma^j]$  has no dominating set of size at most, say  $2k$ , then we reject. This can be done in polynomial time, since DOMINATING SET admits a PTAS on apex-minor free graphs [7]. We thus may assume that there is a dominating set of size  $2k$  in  $G[V_\sigma^j]$ . It is known that an apex-minor free graph with a dominating set of size  $\kappa$  has treewidth  $\mathcal{O}(\sqrt{\kappa})$ , where  $\mathcal{O}$  hides a factor depending on the apex graph whose minors are excluded [6, 9, 12]. We can use a constant-factor approximation algorithm of Demaine [8] to find a tree decomposition  $\mathcal{T}$  of width  $w \in \mathcal{O}(\sqrt{k})$ . Finally, we solve the problem via dynamic programming over the tree decomposition. Bonnet et al. [2] gave a  $\mathcal{O}^*(2^w)$ -time algorithm that solves MAX  $\alpha$ -FCGP with a tree decomposition of width  $w$  given. We need to solve a slightly more general problem because  $\mathcal{T}$  is the tree decomposition is over  $V_\sigma^j$ . To remove  $V \setminus V_\sigma^j$ , we introduce a weight  $\omega: V_\sigma^j \rightarrow \mathbb{N}$  defined by  $\omega(v) = |N(v) \cap (V \setminus V_\sigma^j)|$ . The objective is then to maximize  $\text{cov}_\alpha(C) + \alpha \sum_{v \in C} \omega(v)$ . The dynamic programming algorithm of Bonnet et al. can be adapted to solve this weighted variant in the same running time. Thus, we obtain a  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$ -time algorithm for MAX  $\alpha$ -FCGP.

For MIN  $\alpha$ -FCGP, we can show the following lemma whose proof is omitted because it is almost analogous to the previous one. The only change is that,  $V_\sigma$  refers to the vertices in the non-decreasing order of degrees. Also, we consider the regime where  $0 \leq \alpha \leq 1/3$ , which implies  $\alpha \leq 1 - 2\alpha$ , which would give the reverse inequality in (1).

► **Lemma 6.** *Let  $G = (V_\sigma, E)$  be a yes-instance for MAX  $\alpha$ -FCGP, where  $0 \leq \alpha \leq 1/3$ . Let  $C = \{u_{i_1}, u_{i_2}, \dots, u_{i_k}\}$  be the lexicographically smallest solution for MAX  $\alpha$ -FCGP and  $u_{i_k} = v_j$  for some  $j$ . Then  $C$  is a dominating set of size  $k$  for  $G[V_\sigma^j]$ .*

With this lemma at hand, an analogous algorithm solves MIN  $\alpha$ -FCGP in  $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$  time, thereby proving Theorem 4.

## 4 Conclusion

In this paper, we demonstrated that the algorithms exploiting the “degree-sequence” that have been successful for designing algorithms for MAX  $k$ -VERTEX COVER naturally generalize to MAX/MIN  $\alpha$ -FCGP. Specifically, we designed FPT approximations for MAX/MIN  $\alpha$ -FCGP parameterized by  $k, \alpha$ , and  $\epsilon$ , for any  $\alpha \in (0, 1]$ . For MAX  $\alpha$ -FCGP, this result is tight since, when  $\alpha = 0$ , the problem is equivalent to DENSEST  $k$ -SUBGRAPH, which is hard to approximate in FPT time [21]. We also designed subexponential FPT algorithms for MAX  $\alpha$ -FCGP (resp. MIN  $\alpha$ -FCGP) for the range  $\alpha \geq 1/3$  (resp.  $\alpha \leq 1/3$ ) on any apex-minor closed family of graphs. It is a natural open question whether one can obtain subexponential FPT algorithms for MAX/MIN  $\alpha$ -FCGP for the entire range  $\alpha \in [0, 1]$ . A notable special case is that of DENSEST  $k$ -SUBGRAPH on planar graphs. In this case, the problem is not even known to be NP-hard, if the subgraph is allowed to be disconnected. For the DENSEST CONNECTED  $k$ -SUBGRAPH problem, it was shown by Keil and Brecht [18] that the problem is NP-complete on planar graphs. From the other side, it can be shown that DENSEST CONNECTED  $k$ -SUBGRAPH admits a subexponential in  $k$  randomized algorithm on apex-minor free graphs using the general results of Fomin et al. [13]. Thus, dealing with disconnected dense subgraphs is difficult for both algorithms and lower bounds.

---

## References

- 1 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Implicit Branching and Parameterized Partial Cover Problems. *Journal of Computer and System Sciences*, 77(6):1159–1171, 2011.
- 2 Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Emeric Tourniaire. Multi-parameter Analysis for Local Graph Partitioning Problems: Using Greediness for Parameterization. *Algorithmica*, 71(3):566–580, 2015.
- 3 Leizhen Cai. Parameterized Complexity of Cardinality Constrained Optimization Problems. *The Computer Journal*, 51(1):102–121, 2008.
- 4 Derek G. Corneil and Yehoshua Perl. Clustering and domination in perfect graphs. *Discret. Appl. Math.*, 9(1):27–39, 1984. doi:10.1016/0166-218X(84)90088-X.
- 5 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.
- 6 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional parameters and local treewidth. *SIAM J. Discret. Math.*, 18(3):501–511, 2004. doi:10.1137/S0895480103433410.
- 7 Erik D. Demaine and Mohammad Taghi Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 590–601. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070514>.
- 8 Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 637–646. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.14.

- 9 Erik D. Demaine and MohammadTaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Comb.*, 28(1):19–36, 2008. doi:10.1007/s00493-008-2140-4.
- 10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 11 Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto-Rodriguez, and Frances A. Rosamond. Cutting up is hard to do: the parameterized complexity of  $k$ -cut and related problems. In *Computing: the Australasian Theory Symposium, CATS 2003, Adelaide, SA, Australia, February 4-7, 2003*, volume 78 of *Electronic Notes in Theoretical Computer Science*, pages 209–222. Elsevier, 2003. doi:10.1016/S1571-0661(04)81014-4.
- 12 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Contraction bidimensionality: The accurate picture. In *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 706–717. Springer, 2009. doi:10.1007/978-3-642-04128-0\_63.
- 13 Fedor V. Fomin, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Subexponential parameterized algorithms for planar and apex-minor-free graphs via low treewidth pattern covering. *SIAM J. Comput.*, 51(6):1866–1930, 2022. doi:10.1137/19m1262504.
- 14 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011. doi:10.1016/j.ipl.2011.05.016.
- 15 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized Complexity of Vertex Cover Variants. *Theory of Computing Systems*, 41(3):501–520, 2007.
- 16 Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for  $k$ -cut. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 113–123. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00020.
- 17 Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for  $k$ -cut. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2821–2837. SIAM, 2018. doi:10.1137/1.9781611975031.179.
- 18 J. Mark Keil and Timothy B. Brecht. The complexity of clustering in planar graphs. *J. Combin. Math. Combin. Comput.*, 9:155–159, 1991.
- 19 Tomohiro Koana, Christian Komusiewicz, André Nichterlein, and Frank Sommer. Covering Many (or Few) Edges with  $k$  Vertices in Sparse Graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS '22)*, volume 219 of *LIPICs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 20 Pasin Manurangsi. A Note on Max  $k$ -Vertex Cover: Faster FPT-AS, Smaller Approximate Kernel and Improved Approximation. In *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASICs*, pages 15:1–15:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICs.SOSA.2019.15.
- 21 Pasin Manurangsi, Aviad Rubinfeld, and Tselil Schramm. The strongish planted clique hypothesis and its consequences. In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPICs*, pages 10:1–10:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ITCS.2021.10.
- 22 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.
- 23 Hadas Shachnai and Meirav Zehavi. Parameterized Algorithms for Graph Partitioning Problems. *Theory of Computing Systems*, 61(3):721–738, 2017. doi:10.1007/s00224-016-9706-0.

# Graph Connectivity with Noisy Queries

Dimitris Fotakis   

National Technical University of Athens, Greece

Evangelia Gergatsouli   

University of Wisconsin-Madison, WI, USA

Charilaos Pipis  

National Technical University of Athens, Greece

Miltiadis Stouras  

École Polytechnique Fédérale de Lausanne, Switzerland

Christos Tzamos   

University of Wisconsin-Madison, WI, USA

University of Athens, Greece

---

## Abstract

Graph connectivity is a fundamental combinatorial optimization problem that arises in many practical applications, where usually a spanning subgraph of a network is used for its operation. However, in the real world, links may fail unexpectedly deeming the networks non-operational, while checking whether a link is damaged is costly and possibly erroneous. After an event that has damaged an arbitrary subset of the edges, the network operator must find a spanning tree of the network using non-damaged edges by making as few checks as possible.

Motivated by such questions, we study the problem of finding a spanning tree in a network, when we only have access to noisy queries of the form “Does edge  $e$  exist?”. We design efficient algorithms, even when edges fail adversarially, for all possible error regimes; 2-sided error (where any answer might be erroneous), false positives (where “no” answers are always correct) and false negatives (where “yes” answers are always correct). In the first two regimes we provide efficient algorithms and give matching lower bounds for general graphs. In the False Negative case we design efficient algorithms for large interesting families of graphs (e.g. bounded treewidth, sparse). Using the previous results, we provide tight algorithms for the practically useful family of planar graphs in all error regimes.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** algorithms under uncertainty, graph connectivity, spanning tree, noisy queries, online algorithms, stochastic optimization

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.47

**Related Version** *Full Version:* <https://arxiv.org/abs/2208.10423>

**Funding** This work is partially supported by NTUA Basic Research Grant (PEBE 2020) “Algorithm Design through Learning Theory: Learning-Augmented and Data-Driven Online Algorithms (LEADAlgo)”.

## 1 Introduction

From road and railway networks to electric circuits and computer networks, maintaining an operational spanning subgraph of a network is crucial in many real-life applications. This problem can be viewed as a traditional combinatorial optimization problem, graph connectivity. However, in real-life applications, edges sometimes fail; a natural destruction on the road or a malfunction in the connector cable can render a link in the network non



© Dimitris Fotakis, Evangelia Gergatsouli, Charilaos Pipis, Miltiadis Stouras, and Christos Tzamos; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 47; pp. 47:1–47:14

Leibniz International Proceedings in Informatics



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

operational. Worse still, detecting faulty connections is not always that simple; our tools may be unreliable, resulting in untrustworthy indications. Can we efficiently find an operational spanning tree, when we cannot obtain reliable signals on the operation of edges?

**In this work we study the problem of finding a spanning tree using noisy queries on the existence of edges.** Specifically, given a moldgraph  $G$ , where some subgraph  $H$  of  $G$  is realized, we have access to an oracle that answers questions of the form “Does edge  $e$  exist in  $H$ ?”. The answers the oracle gives are *inconsistent*: for a specific edge  $e$ , the answer might differ every time. Queries to the oracle are costly, therefore our goal is to find a spanning tree asking as few queries as possible. We design efficient algorithms that achieve this goal, in all 3 different error regimes; 2-sided (where any answer might be erroneous), 1-sided false positives (where “no” answers are always correct) and 1-sided false negatives (where “yes” answers are always correct).

## 1.1 Our Contribution

As a warm-up, we begin by solving the simpler problem of learning whether the tree is connected (Theorem 7). Following that, we proceed to our main problem, which is finding a realized spanning tree in three different error regimes. In the 2-sided error case, we give an algorithm performing  $O(m \log m)$  queries and show that no algorithm can perform better, even on special cases like sparse graphs. In the 1-sided False Negative error regime, we design an algorithm that is optimal on planar graphs and yields efficient guarantees for other special families like graphs with treewidth  $k$  or degeneracy  $k$  ( $O(km)$  queries) and graphs with Hadwiger number  $k$  ( $O(k\sqrt{\log km})$  queries). The same algorithm can be used for general graphs, and outside the aforementioned families, its performance gracefully degrades to that of the naive strategy (that is  $O(m \log n)$  queries). It is noteworthy that this is the only algorithm that breaks the  $O(m \log m)$  barrier and it achieves it by employing an adaptive strategy.

In the False-Positive error case, our algorithm obtains tight guarantees for general graphs (Theorem 19), while in the special case where the realized graph is acyclic and the moldgraph is planar, the query complexity becomes linear. Both in the False-Negative and the False-Positive case, our algorithms *do not need to know* whether the graph has any special properties (sparsity, acyclic realized subgraph etc), they can simply run a unified algorithm that achieves the best guarantee according to each case.

Our results, imply tight algorithms for planar graphs, which is a family of graphs that is frequently encountered in road/railway networks, electrical circuits, image processing/computer vision [15, 14].

## 1.2 Related Work

Variations of this problem have been studied in literature, however they differ from our setting. A crucial assumption in all of the previous works, is that each edge has an independent probability of existence, which however, does not always bear out in practice. For example, a power outage may deem several local network links non-operational. In our work, we allow the existence of different edges to be arbitrarily correlated.

Feige et al. in [4] first studied the evaluation of boolean decision trees where the nodes are noisy and the goal is to find the correct leaf within a tolerance parameter  $Q$ . In [6, 5] the authors study finding spanning trees in Erdos-Renyi graphs, a special class of random graphs where each edge exists independently with some probability. In their setting each edge has a *query cost*, and the goal is to find a spanning tree using the least number of queries. However,

contrary to their setting, we handle both adversarially selected realized graphs and noisy answers to existence queries. More recently, Lyu et al. in [11] also studied the problem of finding a MST when the edges can fail with some probability. The work of Hoffmann et al. [7] and Erlebach and Hoffmann [3] considers verifying spanning trees where each edge has a weight inside an uncertainty region and the algorithm can ask the edge to reveal it. The goal is to find a small weight spanning tree without asking all the edges for their weight. None of these settings account for *inconsistent* queries, like in our setting.

In Bhaskara et al. [1] they studied the problem of finding a shortest path in a graph where there are ML-generated hints on the edge lengths, and also costly access to a zero-error oracle. The goal is to find a good enough shortest path, without asking the oracle too many times.

On a different problem in Mazumdar and Saha [12], the authors study clustering in a similar setting to ours where they have access to queries of the form “is  $u$  and  $v$  in the same cluster?”. However in their model, the oracle gives *consistent* answers on each query.

## 2 Preliminaries

In GRAPH CONNECTIVITY WITH NOISY QUERIES we are given a graph  $G = (V, E)$ , with edge set  $E$  and node set  $V$  called the *moldgraph*, then an adversary selects a *connected* subgraph  $G'$  of  $G$  to be realized. The goal of the algorithm is to find a spanning tree in the subgraph  $G'$  spending as little time as possible gathering information. The algorithm does not directly observe the subgraph that is realized, but only has access to an oracle that answers questions of the form “Is edge  $e \in E$  realized?”. Each call to this oracle costs 1, therefore our goal is to find a realized spanning tree, with constant probability using the minimum number of queries to the oracle.

This oracle, however, is noisy and inconsistent; it might not give the correct answer and when asked multiple times on the same edge, it may give different answers. More formally, the oracle is a function  $\mathcal{O} : E \rightarrow \{\text{Yes}, \text{No}\}$ , that given an edge  $e \in E$  answers either *Yes* or *No*, indicating whether the edge is realized or not. We study all the possible error types for the oracle, outlined below.

**2-sided error.** the oracle’s answer is wrong (ie. with “No” for realized edges and “Yes” for non-realized) with *constant* error probability  $p < 1/2$ .

**1-sided error, False Negative (FN).** if the edge is not realized then the response is always “No”. If it is realized then the response is “No” with *constant* error probability  $p < 1/2$ , and “Yes” with probability  $(1 - p)$ . Thus, when the oracle responds “Yes” then it is certain that this edge is realized, but when it responds “No” then the edge may or may not be realized, hence the False Negative responses.

**1-sided error, False Positive (FP).** if the edge is realized then the response is always “Yes”. If it is not realized then the response is “Yes” with *constant* error probability  $p < 1/2$ , and “No” with probability  $(1 - p)$ . Thus, when the oracle responds “No” then it is certain that this edge is not realized, but when it responds “Yes” then the edge may or may not be realized, hence the False Positive responses.

## 2.1 Graphs notation

We present some useful definitions for concepts we use throughout the paper. We begin by our definition of sparsity. Another possible name for the  $\rho$  parameter in the literature is “average degree” of the graph.

► **Definition 1** ( $\rho$ -sparse Graphs). *A graph  $G$  with  $m$  edges and  $n$  vertices is called  $\rho$ -sparse if  $m \leq \rho n$ .*

It is noteworthy that our definition of  $\rho$ -sparsity for graphs is weaker than the usual definitions of graph sparsity, in the sense that it can be easily satisfied without requiring local sparsity properties. This means that constructing efficient algorithms for this property, directly translates to efficient algorithms for other usual sparsity parameters. For example, using our definition,  $k$ -degenerate graphs are  $k$ -sparse [9],  $k$ -treewidth graphs are also  $k$ -sparse [13], and graphs with Hadwiger number  $k$  are  $O(k\sqrt{\log k})$ -sparse [8, 2].

► **Definition 2** (Edge Contraction). *The edge contraction operation on an edge  $e = \{u, v\}$  of a graph  $G$  results in a new graph  $G'$  wherein  $u$  and  $v$  are replaced by a new vertex  $uv$  which is connected to all vertices of  $G$  that were incident to either  $u$  or  $v$ .*

► **Definition 3** (Graph minor [10]). *A graph  $H$  is called a minor of a graph  $G$ , if  $H$  can occur after applying a series of edge deletions, vertex deletions, and edge contractions.*

► **Definition 4** (Minor-closed Graph Family). *A set  $\mathcal{F}$  of graphs is called a minor-closed graph family if for any  $G \in \mathcal{F}$  all minors of  $G$  also belong to  $\mathcal{F}$ .*

The algorithms that we present for FN and FP queries produce and handle multigraphs during intermediate steps. Here, we present and define some of the main concepts we are using from multigraphs.

An edge  $e = \{u, v\}$  in a multigraph is a link connecting  $u$  and  $v$ . There might be other parallel edges between  $u$  and  $v$ , and each one is distinct. Sometimes we will need to work with the set of all parallel edges between  $u$  and  $v$ . We call this set a *super-edge* between  $u$  and  $v$ . For simplicity, we consider simple edges to be super-edges with size 1.

► **Definition 5** (Neighborhood). *For a vertex  $v \in V$ , we denote by  $N(v)$  its neighborhood, which is the set of all super-edges with  $v$  as one of the endpoints.*

► **Definition 6** (Degree). *For a node  $v \in V$ , degree  $\deg(v)$  of  $v$  is the size of  $N(v)$ .*

Finally, we note that whenever an algorithm performs an edge-contraction operation on edge  $e$  of a multigraph, then we delete all the other parallel edges to  $e$  as well, leaving no self-loops in the graph. Moreover, if any parallel super-edges result after the contraction, they are replaced by a larger super-edge, their union.

## 3 Verifying Connectivity

As a warm-up, before attempting to find a realized spanning tree of the graph, we begin by presenting an algorithm for verifying the connectivity of a tree. Specifically, given a tree, which could be the operating spanning tree of a network, we want to verify whether it is connected or not, that is whether all of its edges are realized.

Naively verifying this property (ie. performing a fixed number of queries on every edge) needs  $O(n \log n)$  queries to yield a constant probability guarantee. However, we show that it is possible to verify the connectivity of the tree using only  $O(n)$  queries, which is asymptotically the same as if our oracle had no noise.

■ **Algorithm 1** *Verify*( $T$ ): A verification protocol for tree connectivity with 2-sided error.

**Input:** Tree  $\mathcal{T}$  of  $n$  edges, **Parameters:**  $\varepsilon, \delta, p$

**Output:** True iff  $T$  is connected

---

```

1: threshold  $\leftarrow \lceil \log_{\frac{1-p}{p}} \left( \frac{1}{\delta} \right) \rceil$ 
2: budget  $\leftarrow \lceil \frac{1}{\varepsilon} \cdot \frac{1}{1-2p} \rceil \cdot \textit{threshold} \cdot n$ 
3: for  $e \in E(T)$  do
4:   counter  $\leftarrow 0$ 
5:   while (counter < threshold) & (budget > 0) do
6:     q  $\leftarrow$  Query edge  $e$ 
7:     budget  $\leftarrow$  budget  $-1$ 
8:     if q is “Yes” then
9:       counter  $\leftarrow$  counter  $+1$ 
10:    else
11:      counter  $\leftarrow$  counter  $-1$ 
12:    end if
13:  end while
14:  if budget = 0 then
15:    return False
16:  end if
17: end for
18: return True

```

---

Our algorithm (Algorithm 1) has a predetermined budget of queries that depends on the requested guarantees. It fixes an ordering of the edges, and for each edge it performs queries until it receives  $c$  more positive answers than negative ones (where  $c$  is also fixed and depends on the requested guarantees). For realized edges, since  $p < 1/2$ , the expected number of queries until they reach  $c$  is polynomial in  $c$ . On the other hand, the probability to never yield  $c$  more positive answers than negative ones, within the budget, is exponentially small in  $c$ . Using a global budget instead of a fixed per-edge number of queries, allows us to dynamically allocate it based on need. We save up queries from realized edges that reach the threshold quickly and spend it on other edges that are slow to reach the threshold. On the other hand, the probability that a non-realized edge reaches  $c$  is exponentially small in  $c$ . As a result, if a non-realized edge exists in the tree, the algorithm will consume all of its budget on this edge before reaching the threshold. Using this approach, Algorithm 1 achieves the following guarantees.

► **Theorem 7.** *Algorithm 1 correctly classifies connected trees with probability  $1 - \varepsilon$  and disconnected trees with probability  $1 - \delta$ , while performing at most  $O\left(\frac{1}{\varepsilon} n \log \frac{1}{\delta}\right)$  queries.*

The proof is deferred to the full version. As a Corollary of Theorem 7, we get that, for constant probability guarantees, Algorithm 1 requires  $O(n)$  queries.

► **Corollary 8.** *Algorithm 1 correctly classifies connected trees with probability .99 and disconnected trees with probability .99, using  $O(n)$  queries.*

## 4 Two-sided error oracle

Moving on to our main result, we show how to find a realized spanning tree in the 2-sided error oracle. Recall that in this case, the oracle may give false responses either when the edge is realized or when it is non-realized. The main result for this regime is the following theorem.

► **Theorem 9.** *In the 2-sided error regime, there exists an algorithm that finds a realized spanning tree with high probability, using  $O(m \log m)$  queries in a moldgraph of  $m$  edges. Moreover, no algorithm can do better than  $\Omega(m \log m)$ .*

To show this theorem, we separately show the upper and lower bounds in Lemmas 10 and 12. Combining them we immediately get the theorem.

### 4.1 Upper Bound

In order to show the upper bound (Lemma 10), we describe an algorithm that achieves this query complexity and prove its correctness in Lemma 11

► **Lemma 10.** *In the 2-sided error regime, there exists an algorithm that finds a realized spanning tree with high probability, using  $O(m \log m)$  queries in a moldgraph of  $m$  edges.*

Our algorithm, described below, uses the same idea as the naive algorithm for the connectivity verification problem.

#### Naive algorithm

The algorithm is as follows. First, query each edge of the moldgraph  $O(\log m)$  times. Second, by treating all edges with more “Yes” than “No” responses as realized, compute a spanning tree on the discovered realized subgraph. If the graph is disconnected, output any spanning tree of the moldgraph.

The algorithm performs  $O(m \log m)$  queries in total. The following lemma shows the correctness of the algorithm, and its proof is deferred to the full version.

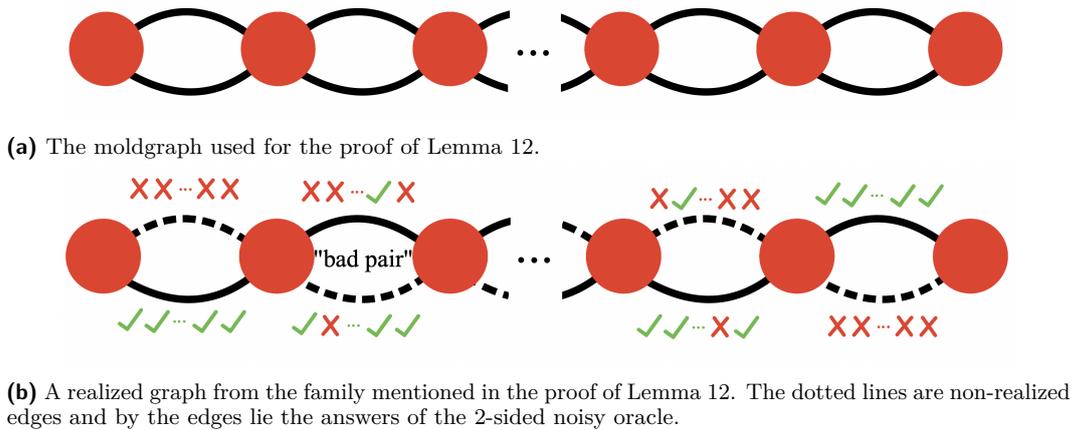
► **Lemma 11.** *The Naive algorithm finds a realized spanning tree with high probability.*

### 4.2 Lower Bound

Switching gears towards the lower bound, we give an instance showing that we cannot hope for anything better than the  $O(m \log m)$  upper bound shown in the previous section. As we expected, this is a strictly harder problem than verifying the connectivity of a tree, since in this case we have to identify a realized edge in each independent cut of the moldgraph.

► **Lemma 12.** *In the 2-sided error regime, there exists a graph where any algorithm requires  $\Omega(m \log m)$  queries to discover a spanning tree with constant failure probability.*

**Proof of Lemma 12.** We consider the graph in Figure 1 which has  $n + 1$  vertices and  $2n$  edges. Between any two consecutive vertices there exist 2 parallel edges, one of which is realized and the other is not. Note that this moldgraph is a multigraph for simplicity of presentation. But the arguments hold analogously for a 2-by- $n$  grid moldgraph, if we additionally give our algorithms the extra information that all vertical edges are realized. This only makes an algorithm more powerful as, in the worst-case, it can choose to ignore this information and execute independently. Then, we show that even with this extra knowledge, no algorithm can achieve a better query complexity than the desired bound.



■ **Figure 1**

Any algorithm on this graph will have to treat each edge-pair independently of the others, because uncovering a realized edge in one pair does not give any information on the solution to other pairs. Moreover, the optimal strategy for any algorithm is to query both edges in a pair an equal number of times and then pick the one with the majority of positive responses. Otherwise, picking the one with the least positive responses gives us smaller probability of success.

Let us suppose that the algorithm performs  $k$  queries on a specific edge-pair. If more than  $k/2$  of them give false responses then the wrong edge will be picked as realized. Consequently, the algorithm will have to perform more than  $k$  queries on this edge-pair to discover the correct edge. We set  $k = \Theta(\log(n/\log n)) = \Theta(\log n)$  and call such a faulty edge-pair a “bad” pair. Then, by the fact that false queries follow a binomial distribution, we get  $\Pr[\text{bad}] > p^{k/2} > \log n/n$ . Thus the probability that *at least* one such “bad pair” exists is

$$\Pr[\text{bad exists}] = 1 - \Pr[\text{all good}] \geq 1 - \left(1 - \frac{\log n}{n}\right)^n \geq 1 - \frac{1}{n}.$$

Assuming that there exists such a “bad pair” then the algorithm will have to perform  $\Omega(\log n)$  queries on it. However, there is no way to know from the beginning which pair it will be. This means that any algorithm in this case will have to pick an  $\alpha$ -fraction of edge-pairs on which to perform these many queries, for some (not necessarily constant)  $\alpha$ . Define  $Q_b$  to be the event that the algorithm performs  $\Omega(\log n)$  queries on the “bad pair”. Then,  $\Pr[Q_b] = \alpha$  and the probability of failure for the algorithm when a bad pair exists is

$$\Pr[\text{fail}|\text{bad exists}] = \alpha \Pr[\text{fail}|Q_b] + (1 - \alpha) \geq (1 - \alpha)$$

Thus, the total probability of failure to discover a spanning tree will be

$$\Pr[\text{fail}] \geq \Pr[\text{fail}|\text{bad exists}] \Pr[\text{bad exists}] \geq (1 - \alpha) \left(1 - \frac{1}{n}\right)$$

If we want the failure probability to be constant then it must hold that  $\alpha = 1 - o(1)$ . Consequently, we perform  $\Omega(\log n)$  queries on  $\alpha n = \Theta(n)$  edges, giving us an  $\Omega(n \log n)$  lower bound for this graph instance. The desired lower bound comes from the fact that any algorithm performing better than  $\Omega(m \log m)$  queries would perform better than  $\Omega(n \log n)$  in this instance, as it has  $m = O(n)$ . ◀

## 5 One-sided error oracle: False Negatives

Moving on to the one-sided error regime, we first consider the case of False Negative errors in the oracle. Our main contribution is an algorithm for the case of sparse and minor-closed graph families. We also show how we can obtain an algorithm for general graphs, that still performs well on  $\rho$ -sparse graphs, even when this property is unknown to us.

Finally, we show how to combine these results to obtain an algorithm for planar graphs that uses  $O(m)$  queries to find a spanning tree, while no algorithm can do better (Corollary 17). This also implies better guarantees for other special families of graphs, as shown in Corollary 18.

► **Theorem 13.** *In the 1-sided False Negative regime, given a moldgraph of  $m$  edges and  $n$  nodes, there exists an algorithm that finds a realized spanning tree using  $O(m \log n)$  queries in expectation. If the moldgraph belongs to a  $\rho$ -sparse and minor-closed family, the algorithm uses  $O(\rho m)$  queries in expectation.*

**Proof of Theorem 13.** To create such an algorithm, we can run in parallel both the Naive Algorithm presented in the next subsection (Lemma 14) and Algorithm 3 for sparse minor-closed graph families. In particular, it suffices to let the two algorithms perform their queries alternately and independently of each other.

If the moldgraph belongs in a  $\rho$ -sparse and minor-closed family, then by Lemma 15, Algorithm 3 will perform  $O(\rho m)$  queries in expectation before finding a spanning tree. For each one of these queries, we have just performed one extra query for the naive algorithm. Thus, eventually the combined algorithm will find the spanning tree in the same query complexity.

Otherwise, in general graphs, by using Lemma 14 for the Naive Algorithm and a similar argument as before, we get  $O(m \log n)$  queries in expectation. ◀

### 5.1 Warm-up: naive algorithm

We start by describing the naive algorithm used for general graphs, that obtains the  $O(m \log n)$  guarantee. The algorithm proceeds in rounds, where in each round it performs one query on all  $m$  edges of the moldgraph. It repeats this process until  $n - 1$  realized edges forming a spanning tree are discovered. Note that in the case of FN queries we can be completely certain when a realized edge is discovered. The proof is deferred to the full version.

► **Lemma 14.** *The Naive Algorithm finds a spanning tree while performing  $O(m \log n)$  queries on expectation.*

### 5.2 An Algorithm for $\rho$ -sparse graphs

We present the algorithm used to obtain better guarantees for the family of  $\rho$ -sparse and minor closed graphs. The algorithm is presented in Algorithm 3 and the guarantees in Lemma 15.

► **Lemma 15.** *Given a moldgraph of  $m$  edges, belonging to a  $\rho$ -sparse and minor-closed family of graphs, Algorithm 3 performs  $O(\rho m)$  queries in expectation and uncovers a realized spanning tree with probability 1.*

The basic primitive that our algorithm uses is a **DISCOVER** subroutine that cleverly orders queries in a way that it allows it to explore multiple edge sets in parallel. The subroutine is presented in more detail in Algorithm 2 and the Lemma that follows.

■ **Algorithm 2** DISCOVER( $\mathcal{S}$ ): Discovers an edge in a collection  $\mathcal{S}$  of edge sets.

**Input:** Set  $\mathcal{S} = \{E_1, E_2, \dots, E_k\}$

**Output:** Realized edge  $e$

---

```

1: while no edge is found do
2:   for  $i = 1$  to  $k$  do
3:     Let  $e$  be the next edge in the cyclic order of  $E_i$ 
4:      $q \leftarrow$  Query edge  $e$ 
5:     if  $q$  is “Yes” then
6:       return  $e$ 
7:     end if
8:   end for
9: end while

```

---

■ **Algorithm 3** SolveSparseFN( $G$ ): Solves the problem in sparse moldgraphs using an 1-sided error oracle with False Negatives.

**Input:** Graph  $G$

**Output:** A realized spanning tree

---

```

1: if  $G$  is a single node then
2:   return  $\emptyset$ 
3: end if
4:  $u = \operatorname{argmin}\{\deg(v)\}$ 
5:  $e = \text{DISCOVER}(N(u))$  // Let  $G'$  be the resulting graph after contracting edge  $e$ .
6: return SolveSparseFN( $G'$ )  $\cup \{e\}$ 

```

---

► **Lemma 16.** *Let  $\mathcal{S}$  be a collection of  $k$  sets of edges. Assume as well that there exists at least one set in  $\mathcal{S}$  containing some realized edge, then Algorithm 2 finds and returns such an edge using at most  $2k|E_f|$  queries in expectation, where  $E_f$  is the set containing the found edge.*

The proof is deferred to the full version. We are now ready to prove the main result of the section, Lemma 15.

**Proof of Lemma 15.** We use induction on the number  $n$  of vertices of  $G$  to prove that Algorithm 3 performs at most  $4\rho m$  queries in expectation. Throughout this proof we denote by  $m$  the total number of simple edges of the graph, and by  $m_s$  the number of super-edges.

The base case of induction for  $n = 1$  holds trivially, as the algorithm needs no queries. Now assume that it holds for all graphs of at most  $n - 1$  vertices. We will prove that it also holds for graphs of  $n$  vertices.

By using the sparsity property that  $m_s \leq \rho n$ , we can see that there always exists at least one vertex  $u$  with degree  $\deg(u) \leq 2\rho$ . Otherwise, the number of super-edges would be

$$m_s = \frac{1}{2} \sum_{v \in V} \deg(v) > \rho n$$

which leads to the graph not being  $\rho$ -sparse, a contradiction.

The algorithm uses Algorithm 2 as a basic subroutine, passing as input the neighborhood  $N(u)$  of the least-degree vertex  $u$ . This is allowed because  $N(u)$  is a cut of the moldgraph and thus satisfies the assumption of Lemma 16 that at least one edge-set contains a realized edge.

The DISCOVER subroutine will find a realized edge inside a super-edge  $E_f$ , spending in expectation at most  $2 \cdot \deg(u)|E_f| \leq 4\rho|E_f|$  queries.

After contracting the super-edge  $E_f$  that contains the realized edge found by the subroutine, the algorithm will continue on the contracted graph  $G'$  that has  $n - 1$  vertices. It is important to note here that the graph  $G'$  will still be  $\rho$ -sparse, because of our assumption that graphs belong in a minor-closed family. By the induction hypothesis, the algorithm will find a realized spanning tree of  $G'$ , by performing at most  $4\rho(m - |E_f|)$  queries in expectation. Therefore, the total number of queries performed will be at most

$$4\rho|E_f| + 4\rho(m - |E_f|) = 4\rho m$$

in expectation, and thus gives us the desired  $O(\rho m)$  upper bound.  $\blacktriangleleft$

► **Corollary 17.** *If the moldgraph is planar, then Algorithm 3 performs  $O(m)$  queries in expectation and succeeds in finding a realized spanning tree with probability 1. Moreover, this is tight and no better algorithm exists for this case.*

**Proof of Corollary 17.** The upper bound follows directly by Lemma 15 and the fact that planar graphs are a minor-closed and 3-sparse family, as  $m \leq 3n - 6$ .

To argue that the algorithm is tight, we remind that in all cases the realized spanning subgraph is chosen arbitrarily, and even in an adversarial way. This means that any algorithm aiming to find a realized spanning tree has to perform at least  $n - 1$  queries, for the  $n - 1$  realized edges of the spanning tree that it discovers. As planar graphs have  $m = \Theta(n)$  the desired lower bound follows.  $\blacktriangleleft$

Apart from planar graphs, as we hinted when defining  $\rho$ -sparsity, an efficient algorithm for  $\rho$ -sparse graphs immediately gives us efficient algorithms for minor-closed families of graphs that are sparse with respect to other widely-used parameters. Based on [13, 8], we can see that the parameters of treewidth and Hadwiger Number define minor-closed graph families. Moreover,  $k$ -treewidth graphs have sparsity  $k$ , while graphs with Hadwiger number  $k$  are  $O(k\sqrt{\log k})$ -sparse. Hence, applying Lemma 15 we get the following Corollary.

► **Corollary 18.** *Algorithm 3 performs  $O(km)$  queries for graphs with treewidth  $k$  and  $O(k\sqrt{\log km})$  queries for graphs with Hadwiger number  $k$ .*

## 6 One-sided error oracle: False Positives

In the 1-sided, False-Positive error case, we initially present an  $O(m \log m)$  algorithm for general graphs, while dropping the complexity to linear when the realized graph is a tree. Each of these guarantees are tight for their respective case. We also show how using one algorithm, we can capture both cases, without actually knowing the structure of the realized graph, and obtain the tight guarantees for each case. Our formal guarantees are formally stated in Theorem 19.

► **Theorem 19.** *In the 1-sided error FP regime, there exists an algorithm that performs  $O(m \log m)$  queries in the general case and finds a realized spanning tree with high probability. Moreover, if the realized subgraph is a tree, the algorithm performs  $O(m)$  queries in expectation and succeeds to find the spanning tree with high probability. These complexities are tight for their respective cases.*

**Proof of Theorem 19.** Similarly to the case of False Negative oracle errors (Lemma 13) we can combine the Naive Algorithm with Algorithm 4, letting them perform queries alternately. When any of the two algorithms halts, the process ends.

Consider the case of general graphs. By Lemma 20 the naive algorithm succeeds in finding a realized spanning tree with high probability by performing  $O(m \log m)$  queries. Thus, the combined algorithm also succeeds with at most  $O(m \log m)$  queries.

Now, we look at the special case where the moldgraph is planar and the realized subgraph is a tree. By Lemma 23, Algorithm 4 performs at most  $O(m)$  queries in expectation. Let  $X$  be the random variable denoting the number of queries performed by Algorithm 4, and let  $E = \{X > m \log_2(m^2)\}$  be the event that  $X$  is more than the queries of the naive algorithm. Using Markov's inequality we get

$$\Pr[E] \leq \frac{\mathbb{E}[X]}{m \log_2(m^2)} \leq O\left(\frac{1}{\log m}\right)$$

Thus, the number  $Q$  of queries performed before the combined algorithm terminates for this case is

$$\mathbb{E}[Q] = \mathbb{E}[Q|\neg E] \Pr[\neg E] + \mathbb{E}[Q|E] \Pr[E] \leq \mathbb{E}[X] + (m \log_2(m^2))O\left(\frac{1}{\log m}\right) \leq O(m)$$

When the algorithm has terminated, there still exists some probability of failure, in the case that both event  $E$  happens and the naive algorithm fails to find a realized spanning tree. By using the failure probability from Lemma 20 and the previously calculated probability of event  $E$  happening, we get

$$\Pr[E \wedge \text{naive algo fails}] = O\left(\frac{1}{m \log m}\right)$$

Thus, the combined algorithm succeeds in this case with high probability as well, and the proof is concluded.  $\blacktriangleleft$

## 6.1 Upper & Lower Bounds

We start by showing, similar to the previous section, how to obtain the upper bound in the case of general graphs. The proof is deferred to the full version.

► **Lemma 20.** *In the 1-sided error FP regime, there exists an algorithm that performs  $O(m \log m)$  queries in the general case and finds a realized spanning tree with high probability.*

Furthermore, we show the lower bound for the general case graphs, stated formally below. Observe that from the construction of this lower bound, no algorithm can do better in planar or sparse graphs. The proof is deferred to the full version.

► **Lemma 21.** *In the regime of 1-sided FP errors, there exists a realized graph with cycles, for which any algorithm requires  $\Omega(m \log m)$  queries to discover a spanning tree with constant failure probability.*

## 6.2 Special Case: Acyclic Realized Graphs

In this section we focus on the special case where the realized subgraph is a tree. This property implies that for each cycle in the moldgraph, at least one edge of the cycle must be non-realized. Since our oracle only commits False Positive errors, the negative answers

## 47:12 Graph Connectivity with Noisy Queries

■ **Algorithm 4**  $\text{SolvePlanarFP}(G)$ : Discovers a realized spanning tree in planar moldgraphs using an 1-sided error oracle with False Positives.

---

**Input:** Planar Graph  $G(V, E)$

**Output:** A realized spanning tree

```
1: // Construct the dual graph of G
2:  $G' \leftarrow \text{dual}(G)$ 
3: // Inverse all the oracle answers and use SolveSparseFN
4: return  $E(G) \setminus \text{SolveSparseFN}(G')$ 
```

---

are definitive, meaning that we can completely remove the edges which have yielded such answers. We design an algorithm (Algorithm 4) that utilizes this fact and progressively removes non-realized edges until it is left with a tree which will also be the realized subgraph. This result is stated formally in the theorem below.

► **Theorem 22.** *In the regime of 1-sided FP errors, if the moldgraph  $G$  is a planar graph and the realized subgraph is a tree, Algorithm 4 recovers the realized spanning tree with probability 1 while performing at most  $O(m)$  queries in expectation.*

Our main observation is that the FP setting is very similar, and in fact can be reduced to, the FN setting. In the FN setting, we know that at least one realized edge lies in every cut of the moldgraph and we repeatedly query the edges of a cut in order to unveil the realized edge. In a similar manner, in the FP setting, we can repeatedly query the edges of the cycle in order to discover the non-realized edge in every cycle. Hence, the question that arises is: how can one decide a sequence of cycles to query such that the expected number of queries is minimized? Algorithm 4 does so by utilizing the planarity of the moldgraph to consistently find small cycles to query.

### Description of Algorithm 4

Our algorithm reduces the FP setting of this special case to solving an FN instance on the *dual graph* of the moldgraph. The dual of a planar graph has one node for each face of the planar graph, as illustrated in a Figure found in the full version. Two face nodes are connected by an edge in the dual graph if and only if they share an edge in the planar graph. With this transformation we can now focus on cuts of the dual graph which directly correspond to cycles of the moldgraph. Notice that removing an edge in the moldgraph (after identifying it as non-realized) is the same as contracting it in the dual graph. As a result, removing all the non-realized edges until we are left with a tree in the moldgraph is exactly equivalent to contracting the same edges in the dual until we are left with one node (Lemma 23). For this reason, Algorithm 4 constructs the dual graph of the moldgraph and calls Algorithm 3 to retrieve a non-realized spanning tree of the dual graph. Note that we reverse all the oracle's answers before we hand them out to Algorithm 3, since by design it contracts the edges for which we get positive answers but in the FP setting we want to contract the edges of the dual graph whose corresponding edges yielded negative answers.

We will now prove the correctness of Algorithm 4 and bound its query complexity.

► **Lemma 23.** *Let  $G$  be a planar moldgraph,  $T$  be the realized tree of  $G$  and  $G'$  be a dual graph of  $G$ . The induced subgraph of  $G'$  containing all the edges corresponding to non-realized edges of  $G$ , is a spanning tree of  $G'$ .*

The proof of Lemma 23 is deferred to the full version.

**Proof of Theorem 22.** Algorithm 4 constructs the dual graph of  $G$  and uses Algorithm 3 to find a spanning tree of  $G'$  consisting of non-realized edges. From Lemma 23 we know that by removing these edges, we are left with the realized tree  $T$  of  $G$ . All the queries performed during this process are made through Algorithm 3. Therefore, from Lemma 15, the total number of queries conducted is  $O(m)$ . ◀

---

## References

- 1 Aditya Bhaskara, Sreenivas Gollapudi, Kostas Kollias, and Kamesh Munagala. Adaptive probing policies for shortest path routing. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/62da5a6d47be0029801ba74a17e47e1a-Abstract.html>.
- 2 Béla Bollobás, Paul A Catlin, and Paul Erdős. Hadwiger's conjecture is true for almost every graph. *Eur. J. Comb.*, 1(3):195–199, 1980.
- 3 Thomas Erlebach and Michael Hoffmann. Minimum spanning tree verification under uncertainty. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science – 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2014. doi:10.1007/978-3-319-12340-0\_14.
- 4 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.
- 5 Luoyi Fu, Xinzhe Fu, Zhiying Xu, Qianyang Peng, Xinbing Wang, and Songwu Lu. Determining source-destination connectivity in uncertain networks: Modeling and solutions. *IEEE/ACM Trans. Netw.*, 25(6):3237–3252, 2017. doi:10.1109/TNET.2017.2725905.
- 6 Luoyi Fu, Xinbing Wang, and P. R. Kumar. Optimal determination of source-destination connectivity in random graphs. In Jie Wu, Xiuzhen Cheng, Xiang-Yang Li, and Saswati Sarkar, editors, *The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014*, pages 205–214. ACM, 2014. doi:10.1145/2632951.2632990.
- 7 Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPICs*, pages 277–288. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPICs.STACS.2008.1358.
- 8 Alexandr V. Kostochka. Lower bound of the hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984.
- 9 Don R. Lick and Arthur T. White.  $k$ -degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. doi:10.4153/CJM-1970-125-1.
- 10 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.
- 11 Jianhua Lyu, Yiran Ren, Zeeshan Abbas, and Baili Zhang. Reliable route selection for wireless sensor networks with connection failure uncertainties. *Sensors*, 21(21):7254, 2021. doi:10.3390/s21217254.
- 12 Arya Mazumdar and Barna Saha. Clustering with noisy queries. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: <https://proceedings.neurips.cc/paper/2017/file/db5cea26ca37aa09e5365f3e7f5dd9eb-Paper.pdf>.

- 13 Neil Robertson and Paul D Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.
- 14 Frank R. Schmidt, Eno Töppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 351–356. IEEE Computer Society, 2009. doi:10.1109/CVPR.2009.5206863.
- 15 Julian Yarkony and Charless C. Fowlkes. Planar ultrametrics for image segmentation. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 64–72, 2015. URL: <https://proceedings.neurips.cc/paper/2015/hash/3416a75f4cea9109507cacd8e2f2aefc-Abstract.html>.

# Positive Data Languages

Florian Frank  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Stefan Milius  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Henning Urbat  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

---

## Abstract

Positive data languages are languages over an infinite alphabet closed under possibly non-injective renamings of data values. Informally, they model properties of data words expressible by assertions about equality, but not inequality, of data values occurring in the word. We investigate the class of positive data languages recognizable by nondeterministic orbit-finite nominal automata, an abstract form of register automata introduced by Bojańczyk, Klin, and Lasota. As our main contribution we provide a number of equivalent characterizations of that class in terms of positive register automata, monadic second-order logic with positive equality tests, and finitely presentable nondeterministic automata in the categories of nominal renaming sets and of presheaves over finite sets.

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

**Keywords and phrases** Data Languages, Register Automata, MSO, Nominal Sets, Presheaves

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.48

**Related Version** *Full Version with omitted details and proofs:* <https://arxiv.org/abs/2304.12947>

**Funding** *Florian Frank:* Supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393541319/GRK2475/1-2019).

*Stefan Milius:* Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 419850228.

*Henning Urbat:* Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 470467389.

**Acknowledgements** The authors wish to thank Bartek Klin for pointing out the example in Remark 2.10.

## 1 Introduction

Automata over infinite alphabets provide a simple computational model for reasoning about structures involving *data* such as nonces [21], URLs [3], or values in XML documents [25]. Consider, for instance, the (infinite) set  $\mathbb{A}$  of admissible user IDs for a server. The sequence of all user logins within a given time period then forms a finite word  $a_1 \cdots a_n \in \mathbb{A}^*$  over the infinite alphabet  $\mathbb{A}$ , and behaviour patterns may be modelled as data languages over  $\mathbb{A}$ , e.g.

$$\begin{aligned} L_0 &= \{ a_1 \cdots a_n \in \mathbb{A}^* \mid a_i \neq a_n \text{ for all } i < n \} && \text{ (“last user has not logged in before”)}, \\ L_1 &= \{ a_1 \cdots a_n \in \mathbb{A}^* \mid a_i = a_j \text{ for some } i \neq j \} && \text{ (“some user has logged in twice”)}. \end{aligned}$$

Both  $L_0$  and  $L_1$  involve assertions about equality, or inequality, of data values (here, user IDs). However, asserting *inequality* is sometimes considered problematic and thus undesired. For example, since users may have multiple IDs, a logfile  $a_1 \cdots a_n \in L_0$  does not actually guarantee that the last user has not logged in before. In contrast, if  $a_1 \cdots a_n \in L_1$ , then it is guaranteed that some user has indeed logged in twice. The structural difference between the



© Florian Frank, Stefan Milius, and Henning Urbat;  
licensed under Creative Commons License CC-BY 4.0

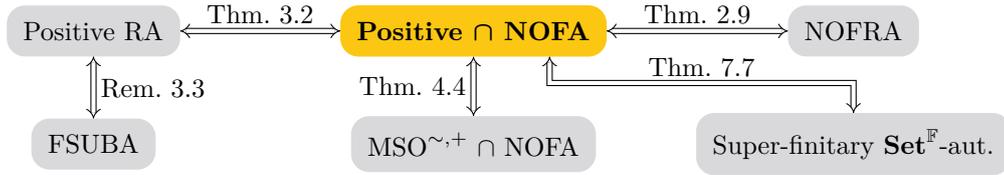
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 48; pp. 48:1–48:15

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Equivalent characterizations of positive NOFA-recognizable languages.

two languages is that  $L_1$  is closed under arbitrary renamings  $\rho: \mathbb{A} \rightarrow \mathbb{A}$  (i.e.  $a_1 \cdots a_n \in L_1$  implies  $\rho(a_1) \cdots \rho(a_n) \in L_1$ ), taking into account possible identification of data values, while  $L_0$  is only closed under injective (equivalently bijective) renamings. We refer to languages with the former, stronger closure property as *positive data languages*. Intuitively, such languages model properties of data words expressible by positive statements about equality of data values. It is one of the goals of our paper to turn this into a theorem.

For that purpose, we build on the abstract account of data languages and their automata based on the theory of nominal sets [13, 27], initiated by the work of Bojańczyk, Klin, and Lasota [6]. Specifically, we investigate *nondeterministic orbit-finite nominal automata (NOFA)*, the nominal version of classical nondeterministic finite automata. We approach the class of *positive* NOFA-recognizable data languages from several different perspectives, ranging from concrete to more abstract and conceptual, and establish the equivalent characterizations summarized in Figure 1. In more detail, our main contributions are as follows.

**Register automata.** NOFAs are known to be expressively equivalent to register automata [17, 19], i.e. finite automata that can memorize data values using a fixed number of registers and test the input for (in)equality with previously stored values. Restricting transitions to positive equality tests leads to *positive register automata*, which correspond to *finite-state unification-based automata (FSUBA)* [18, 32] and are shown to capture precisely positive NOFA-recognizable languages (Theorem 3.2 and Remark 3.3). On the way, we isolate a remarkable property of this language class: while NOFAs generally require the ability to guess data values during the computation to reach their full expressive strength, guessing and non-guessing NOFA are equivalent for positive data languages (Theorem 2.17).

**Monadic second-order logic.** As illustrated above, positive data languages model (only) positive assertions about the equality of data values. To substantiate this intuition, we employ monadic second-order logic ( $\text{MSO}^{\sim}$ ) over data words [4, 9, 25], an extension of classical MSO with equality tests for data values, and consider its restriction  $\text{MSO}^{\sim,+}$  to positive equality tests. While this logic is more expressive than NOFA, we show that within the class of NOFA-recognizable languages it models exactly the positive languages (Theorem 4.4).

**Categorical perspective.** The classical notion of nondeterministic finite automata can be categorified by replacing the finite set of states with a finitely presentable object of a category  $\mathcal{C}$ . For example, NOFAs are precisely nondeterministic  $\mathcal{C}$ -automata for  $\mathcal{C} =$  nominal sets. Apart from the latter category, several other toposes have been proposed as abstract foundations for reasoning about names (data values), most prominently the category of *nominal renaming sets* [12], the category  $\mathbf{Set}^{\mathbb{I}}$  of presheaves over finite sets and injective maps [31], and the category  $\mathbf{Set}^{\mathbb{F}}$  of presheaves over finite sets and all maps (equivalently, finitary set functors) [10]. It is thus natural to study nondeterministic automata in the latter three categories, viz. *nondeterministic orbit-finite renaming automata (NOFRA)*,

*nondeterministic super-finitary*  $\mathbf{Set}^{\mathbb{I}}$ -automata and *nondeterministic super-finitary*  $\mathbf{Set}^{\mathbb{F}}$ -automata. Our final contribution is a classification of their expressive power: we show that  $\mathbf{Set}^{\mathbb{I}}$ -automata are equivalent to NOFAs, while both NOFRAs and  $\mathbf{Set}^{\mathbb{F}}$ -automata capture positive NOFA-recognizable languages (Theorems 2.9 and 7.7). Hence, both nominal and presheaf-based automata are able to recognize positive and all NOFA-recognizable languages, respectively.

## 2 Nominal Automata and Positive Data Languages

For the remainder of the article, we fix a countably infinite set  $\mathbb{A}$  of *data values*, a.k.a. *names* or *atoms*. The goal is to study positive data languages, that is, languages of finite words over  $\mathbb{A}$  closed under arbitrary renamings. This is achieved via the framework of nominal (renaming) sets [12, 13, 27].

### 2.1 Nominal Sets and Nominal Renaming Sets

A *renaming* is a finite map  $\rho: \mathbb{A} \rightarrow \mathbb{A}$ , that is,  $\rho(a) = a$  for all but finitely many  $a \in \mathbb{A}$ . We let  $\text{Fin}(\mathbb{A})$  denote the monoid of renamings, with multiplication given by composition, and  $\text{Perm}(\mathbb{A})$  its subgroup given by *finite permutations*, i.e. bijective renamings. For  $M \in \{\text{Perm}(\mathbb{A}), \text{Fin}(\mathbb{A})\}$  an *M-set* is a set  $X$  equipped with a monoid action  $M \times X \rightarrow X$ , denoted  $(\rho, x) \mapsto \rho \cdot x$ . A subset  $S \subseteq \mathbb{A}$  is a *support* of  $x \in X$  if for every  $\rho, \sigma \in M$  such that  $\rho|_S = \sigma|_S$  one has  $\rho \cdot x = \sigma \cdot x$ . Informally, consider  $X$  as a set of syntactic objects (e.g. words, trees,  $\lambda$ -terms) whose description may involve free names from  $S$ . A *nominal M-set* is an *M-set* where every element  $x$  has a finite support. This implies that  $x$  has a least finite support  $\text{supp } x \subseteq \mathbb{A}$ . A name  $a \in \mathbb{A}$  is *fresh* for  $x$ , denoted  $a \# x$ , if  $a \notin \text{supp } x$ .

Nominal  $\text{Perm}(\mathbb{A})$ -sets are called *nominal sets*, and nominal  $\text{Fin}(\mathbb{A})$ -sets are called *nominal renaming sets*. A nominal renaming set  $X$  can be regarded as a nominal set by restricting its  $\text{Fin}(\mathbb{A})$ -action to a  $\text{Perm}(\mathbb{A})$ -action. The least supports of an element  $x \in X$  w.r.t. both actions coincide [11, Thm. 4.8], so the notation  $\text{supp } x$  is unambiguous.

A subset  $X$  of a nominal *M-set*  $Y$  is *M-equivariant* if  $\rho \cdot x \in X$  for all  $x \in X$  and  $\rho \in M$ . More generally, a map  $f: X \rightarrow Y$  between nominal *M-sets* is *M-equivariant* if  $f(\rho \cdot x) = \rho \cdot f(x)$  for all  $x \in X$  and  $\rho \in M$ . This implies  $\text{supp } f(x) \subseteq \text{supp } x$  for all  $x \in X$ .

We write  $X \times Y$  for the cartesian product of nominal *M-sets* with componentwise action, and  $\coprod_{i \in I} X_i$  for the coproduct (disjoint union) with action inherited from the summands.

Given a nominal set  $X$ , the *orbit* of an element  $x \in X$  is the set  $\{\pi \cdot x : \pi \in \text{Perm}(\mathbb{A})\}$ . The orbits form a partition of  $X$ . A nominal set is *orbit-finite* if it has only finitely many orbits. A nominal renaming set is *orbit-finite* if it is orbit-finite as a nominal set.

► **Example 2.1.** The set  $\mathbb{A}$  with the  $\text{Fin}(\mathbb{A})$ -action  $\rho \cdot a = \rho(a)$  is a nominal renaming set, as is the set  $\mathbb{A}^*$  of finite words over  $\mathbb{A}$  with  $\rho \cdot w = \rho^*(w) = \rho(a_1) \cdots \rho(a_n)$  for  $w = a_1 \cdots a_n$ . The least support of  $a_1 \cdots a_n \in \mathbb{A}^*$  is the set  $\{a_1, \dots, a_n\}$ . The set  $\mathbb{A}^*$  has infinitely many orbits; its equivariant subsets  $\mathbb{A}^n$  (words of a fixed length  $n$ ) are orbit-finite. For instance,  $\mathbb{A}^2$  has the two orbits  $\{aa : a \in \mathbb{A}\}$  and  $\{ab : a \neq b \in \mathbb{A}\}$ . An example of a nominal set that is not a renaming set is  $\mathbb{A}^{\#n} = \{a_1 \dots a_n : a_i \neq a_j \text{ for } i \neq j\}$  with pointwise  $\text{Perm}(\mathbb{A})$ -action.

A nominal set  $X$  is *strong* if, for every  $x \in X$  and  $\pi \in \text{Perm}(\mathbb{A})$ , one has  $\pi \cdot x = x$  if and only if  $\pi$  fixes every element of  $\text{supp}(x)$ . (The “if” statement holds in every nominal set.) For instance, the nominal sets  $\mathbb{A}^{\#n}$ ,  $\mathbb{A}^n$  and  $\mathbb{A}^*$  are strong. Up to isomorphism, (orbit-finite) strong nominal sets are precisely (finite) coproducts  $\coprod_{i \in I} \mathbb{A}^{\#n_i}$  where  $n_i \in \mathbb{N}$ . For every orbit-finite nominal set  $X$ , there exists a surjective  $\text{Perm}(\mathbb{A})$ -equivariant map  $e: Y \rightarrow X$  for

some orbit-finite strong nominal set  $Y$  (see e.g. [23, Cor. B.27]). In fact, if  $o$  is the number of orbits of  $X$ , one may take  $Y = J \times \mathbb{A}^{\#n}$  where  $J = \{1, \dots, o\}$  and  $n = \max_{x \in X} |\text{supp } x|$ . We refer the reader to [14, Sec. 4.1] and [6, Thm. 10.9] for more details on representing orbit-finite nominal sets.

## 2.2 Nominal Automata and Nominal Renaming Automata

The object of interest in this paper is data languages  $L \subseteq \mathbb{A}^*$  closed under renamings:

► **Definition 2.2.**

1. A data language  $L \subseteq \mathbb{A}^*$  is *positive* if it is  $\text{Fin}(\mathbb{A})$ -equivariant.
2. The *positive closure* of  $L \subseteq \mathbb{A}^*$  is given by  $\bar{L} = \{\rho^*(w) : w \in L, \rho \in \text{Fin}(\mathbb{A})\}$ .

A natural automata model for data languages is given by nondeterministic orbit-finite automata [6] over nominal sets and their restriction to nominal renaming sets:

► **Definition 2.3.** Let  $M \in \{\text{Perm}(\mathbb{A}), \text{Fin}(\mathbb{A})\}$ .

1. A *nondeterministic orbit-finite  $M$ -automaton*  $A = (Q, \delta, I, F)$  consists of an orbit-finite nominal  $M$ -set  $Q$  of states, an  $M$ -equivariant transition relation  $\delta \subseteq Q \times \mathbb{A} \times Q$ , and  $M$ -equivariant subsets  $I, F \subseteq Q$  of initial and final states. Nominal orbit-finite  $M$ -automata are called *nondeterministic orbit-finite automata (NOFA)* for  $M = \text{Perm}(\mathbb{A})$  and *nondeterministic orbit-finite renaming automata (NOFRA)* for  $M = \text{Fin}(\mathbb{A})$ .
2. Given a nominal orbit-finite  $M$ -automaton  $A$ , we write  $q \xrightarrow{a} q'$  if  $q' \in \delta(q, a)$ . A *run* of  $A$  on input  $w = a_1 \cdots a_n \in \mathbb{A}^*$  is a sequence  $(q_0, a_1, q_1, a_2, \dots, a_n, q_n)$  such that  $q_0 \in I$  and  $q_r \xrightarrow{a_{r+1}} q_{r+1}$  for  $0 \leq r < n$ . The run is *accepting* if  $q_n \in F$ . The automaton  $A$  *accepts* the word  $w$  if  $A$  admits an accepting run on input  $w$ . The *accepted language*  $L(A) \subseteq \mathbb{A}^*$  is the set of all accepted words. A data language is *NOF(R)A-recognizable* if some NOF(R)A accepts it.

For example, the languages  $L_0$  and  $L_1$  from the Introduction are NOFA-recognizable.

► **Remark 2.4.**

1. The restriction to the input alphabet  $\mathbb{A}$  is for simplicity: all our results extend to alphabets  $\Sigma = \Sigma_0 \times \mathbb{A}$  for a finite set  $\Sigma_0$ , i.e. finite coproducts of copies of  $\mathbb{A}$ .
2. Another use of nominal renaming sets in automata theory appears in the work by Moerman and Rot [24] on deterministic nominal automata with outputs. The restrictions of their model make it unsuitable for language recognition [24, Rem. 4.1] but allow for a succinct representation of computed maps via *separating automata*.

To relate the expressive power of NOFA and NOFRA, we start with a simple observation:

► **Proposition 2.5.** *Every NOFRA accepts a positive language.*

The converse (Theorem 2.9) needs an automata-theoretic construction of the closure of a language. To this end, we first turn the states of a NOFA into a sort of normal form.

► **Remark 2.6** (cf. [6]). Every NOFA  $A = (Q, \delta, I, F)$  is equivalent to one whose nominal set of states is of the form  $J \times \mathbb{A}^{\#m}$  for some finite set  $J$  and  $m \in \mathbb{N}$ . Indeed, choose a nominal set  $Q' = J \times \mathbb{A}^{\#m}$  and an equivariant surjection  $e: Q' \twoheadrightarrow Q$  (see Section 2.1), and consider the NOFA  $A' = (Q', \delta', I', F')$  whose structure is given by the preimages

$$\delta' = (e \times \text{id}_{\mathbb{A}} \times e)^{-1}[\delta], \quad I' = e^{-1}[I], \quad F' = e^{-1}[F].$$

It is not difficult to verify that  $L(A') = L(A)$ ; see also Proposition 6.9. Note that in a NOFA with states  $J \times \mathbb{A}^{\#m}$ , the equivariant sets of initial and final states are of the form  $I = J_I \times \mathbb{A}^{\#m}$  and  $F = J_F \times \mathbb{A}^{\#m}$  for some  $J_I, J_F \subseteq J$ .

► **Construction 2.7** (Positive Closure of a NOFA). Let  $A = (Q, \delta, I, F)$  be a NOFA with states  $Q = J \times \mathbb{A}^{\#m}$  (cf. Remark 2.6). The NOFRA  $\bar{A} = (\bar{Q}, \bar{\delta}, \bar{I}, \bar{F})$  is given by the states  $\bar{Q} = J \times \mathbb{A}^m$ , initial states  $\bar{I} = J_I \times \mathbb{A}^m$ , final states  $\bar{F} = J_F \times \mathbb{A}^m$ , and transitions

$$\bar{\delta} = \{ (j, \rho^* p) \xrightarrow{\rho a} (j', \rho^* p') : (j, p) \xrightarrow{a} (j', p') \text{ in } A \text{ and } \rho \in \text{Fin}(\mathbb{A}) \}.$$

► **Proposition 2.8.** *The NOFRA  $\bar{A}$  accepts the positive closure of the language of  $A$ .*

The proof of  $L(\bar{A}) \subseteq \overline{L(A)}$  is slightly subtle since the transitions of a run in  $\bar{A}$  may be induced by different  $\rho$ 's; some bookkeeping and sensible choice of fresh names ensures compatibility.

Now we come to our first characterization of positive NOFA-recognizable languages:

► **Theorem 2.9.** *A language is positive and NOFA-recognizable iff it is NOFRA-recognizable.*

Indeed, the “if” direction holds due to Proposition 2.5 and because every NOFRA is a NOFA. The “only if” direction follows from Proposition 2.8, using that  $\bar{L} = L$  for positive  $L$ .

► **Remark 2.10.** A NOF(R)A is *deterministic*, and hence called a *DOF(R)A*, if it admits a single initial state and its transition relation is a function  $\delta: Q \times \mathbb{A} \rightarrow Q$ . In contrast to classical finite automata, DOFAs are less expressive than NOFAs [6]. We leave it as an open problem whether Theorem 2.9 restricts to DOF(R)As. In this regard, observe that Construction 2.7 produces a *nondeterministic* automaton  $\bar{A}$  even if the given automaton  $A$  is deterministic. Computing the positive closure of a DOFA-recognizable language necessarily requires the introduction of nondeterminism, as illustrated by the following example due to Bartek Klin (personal communication). Consider the language  $L$  consisting of all words whose last letter appears immediately before the last occurrence of a repeated letter; that is, words of the form  $vabw$  where (i)  $v, w \in \mathbb{A}^*$  and  $a, b \in \mathbb{A}$ , (ii) any two consecutive letters in  $w$  are distinct, (iii) the first letter of  $w$  is distinct from  $b$  and (iv) the last letter of  $w$  is distinct from  $a$ . This language is recognizable by a DOFA, in fact by an orbit-finite nominal monoid [4]. Its positive closure  $\bar{L}$  consists of all words whose last letter appears immediately before *some* occurrence of a repeated letter, which is not DOFA-recognizable.

### 2.3 Abstract Transitions and Runs

Sections 3 and 4 will relate positive NOFA-recognizable languages to register automata and monadic second-order logic. This relies on a presentation of transitions of  $\bar{A}$  in terms of abstract transitions, given by equations involving register entries and input values.

► **Definition 2.11.** Let  $A = (Q, \delta, I, F)$  and  $\bar{A} = (\bar{Q}, \bar{\delta}, \bar{I}, \bar{F})$  be as in Construction 2.7.

1. An *equation* is an expression of the form  $k = \bullet$ ,  $\bullet = k$  or  $k = \bar{k}$ , where  $k, \bar{k} \in \{1, \dots, m\}$ .
2. An *abstract transition* is a triple  $(j, E, j')$  where  $j, j' \in J$  and  $E$  is a set of equations.
3. Every triple  $((j, p), a, (j', p')) \in Q \times \mathbb{A} \times Q$  induces an abstract transition  $(j, E, j')$  defined as follows for  $k, \bar{k} \in \{1, \dots, m\}$  (we write  $(-)_i$  for the  $i$ -th letter of a word):

$$k = \bullet \in E \iff p_k = a, \quad \bullet = k \in E \iff a = p'_k, \quad k = \bar{k} \in E \iff p_k = p'_{\bar{k}}.$$

We let  $\text{abs}(\delta)$  denote the set of abstract transitions induced by transitions in  $\delta$ , and we write  $j \xrightarrow{E} j'$  for  $(j, E, j') \in \text{abs}(\delta)$ .

4. A triple  $((j, q), b, (j', q')) \in \bar{Q} \times \mathbb{A} \times \bar{Q}$  is *consistent* with the abstract transition  $(j, E, j')$  if for every  $k, \bar{k} \in \{1, \dots, m\}$  the following conditions hold:

$$k = \bullet \in E \implies q_k = b, \quad \bullet = k \in E \implies b = q'_k, \quad k = \bar{k} \in E \implies q_k = q'_{\bar{k}}.$$

► **Proposition 2.12.** *For every triple  $((j, q), b, (j', q')) \in \bar{Q} \times \mathbb{A} \times \bar{Q}$ , we have*

$$(j, q) \xrightarrow{b} (j', q') \text{ in } \bar{A} \text{ iff } ((j, q), b, (j', q')) \text{ is consistent with some } (j, E, j') \in \text{abs}(\delta).$$

► **Definition 2.13.** An *abstract run* in  $\bar{A}$  is a sequence  $(j_0, E_1, j_1, E_2, j_2, \dots, E_n, j_n)$  such that  $j_0 \in J_I$  and  $j_{r-1} \xrightarrow{E_r} j_r$  for  $r = 1, \dots, n$ . It is *accepting* if  $j_n \in J_F$ .

► **Notation 2.14.** Given an abstract run  $(j_0, E_1, j_1, E_2, j_2, \dots, E_n, j_n)$ , we inductively define the predicates  $\text{Eq}_k^{(i)}$  ( $i \in \{1, \dots, n\}$ ,  $k \in \{1, \dots, m\}$ ) on the set  $\{1, \dots, n\}$ :

1. if  $\bullet = k$  in  $E_i$  then  $\text{Eq}_k^{(i)}(i)$ ;
2. if  $r < n$  and  $k = \bar{k}$  in  $E_{r+1}$  and  $\text{Eq}_{\bar{k}}^{(i)}(r)$  then  $\text{Eq}_k^{(i)}(r+1)$ .

Informally,  $\text{Eq}_k^{(i)}(r)$  asserts that  $1 \leq i \leq r \leq n$  and that in every run in  $\bar{A}$  of length  $r$  whose transitions are consistent with  $E_1, \dots, E_r$ , the  $i$ -th input letter equals the content of register  $k$  after  $r$  steps. The accepted language may be characterized using these predicates:

► **Proposition 2.15.** *The NOFRA  $\bar{A}$  accepts the word  $b_1 \dots b_n \in \mathbb{A}^*$  iff there exists an accepting abstract run of length  $n$  (with induced predicates  $\text{Eq}_k^{(i)}$ ) such that for  $i, r \in \{1, \dots, n\}$ ,*

$$r < n \text{ and } k = \bullet \text{ in } E_{r+1} \text{ and } \text{Eq}_k^{(i)}(r) \text{ for some } k \implies b_i = b_{r+1}. \quad (2.1)$$

As a first application of this result, we identify an important difference between NOFA and NOFRA concerning the power of guessing data values during the computation:

► **Definition 2.16.** A NOFA/NOFRA is *non-guessing* if each initial state has empty support and for each transition  $q \xrightarrow{a} q'$  one has  $\text{supp } q' \subseteq \text{supp } q \cup \{a\}$ .

The NOFA-recognizable language  $L_0$  from the Introduction is not recognizable by any non-guessing NOFA [17, Ex. 1]. Note that  $L_0$  is not positive; in fact, it is necessarily so, since for positive languages guessing does not add to the expressive power of automata:

► **Theorem 2.17.** *Every positive NOFA-recognizable language is accepted by some non-guessing NOFRA, hence by some non-guessing NOFA.*

To make a NOFRA non-guessing, one keeps track (via the state) of those registers containing data values forced by abstract transitions. The other registers then may be modified arbitrarily, which allows the elimination of guessing transitions.

### 3 Positive Register Automata

We now relate positive NOFA-recognizable languages to register automata, a.k.a. finite-memory automata, originally introduced by Kaminski and Francez [17]; we follow the equivalent presentation by Bojańczyk et al. [6]. A *register automaton* is a quintuple  $A = (C, m, \delta, I, F)$  where  $C$  is a finite set of control states,  $m \in \mathbb{N}$  is the number of registers (numbered from 1 to  $m$ ),  $I, F \subseteq C$  are sets of initial and final states, and  $\delta \subseteq C \times \text{Bool}(\Phi) \times C$  is the set of transitions. Here,  $\text{Bool}(\Phi)$  denotes the set of boolean formulas over the atoms  $\Phi = (\{1, \dots, m\} \times \{\text{before}\} \cup \{\bullet\} \cup \{1, \dots, m\} \times \{\text{after}\})^2$ . Elements of  $\Phi$  are called *equations*; we write  $x = y$  for  $(x, y) \in \Phi$ . Moreover, we denote  $(c, \varphi, c') \in \delta$  by  $c \xrightarrow{\varphi} c'$ . A

configuration of  $A$  is a pair  $(c, r)$  of a state  $c \in C$  and a word  $r \in (\mathbb{A} \cup \{\perp\})^m$  corresponding to a partial assignment of data values to the registers. The initial configurations are  $(c, \perp^m)$  for  $c \in I$ . Given an input  $a \in \mathbb{A}$  and configurations  $(c, r), (c', r')$  we write  $(c, r) \xrightarrow{a} (c', r')$  if this move is consistent with some transition  $c \xrightarrow{\varphi} c'$ , that is, the formula  $\varphi$  is true under the assignment making an atom  $x = y \in \Phi$  true iff the corresponding data values are defined and equal. For instance,  $(k, \text{before}) = \bullet$  is true iff  $r_k \neq \perp$  and  $r_k = a$ , and  $(k, \text{before}) = (\bar{k}, \text{after})$  is true iff  $r_k, r'_k \neq \perp$  and  $r_k = r'_k$ . A word  $w = a_1 \dots a_n \in \mathbb{A}^*$  is *accepted* by  $A$  if it admits an accepting run, viz. a sequence of moves  $(c_0, r_0) \xrightarrow{a_1} (c_1, r_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (c_n, r_n)$  where  $(c_0, r_0)$  is initial and  $c_n \in F$ . The *accepted language*  $L(A) \subseteq \mathbb{A}^*$  is the set of accepted words.

As shown by Bojańczyk et al. [6], register automata accept the same languages as NOFAs. To capture positive languages, we restrict to register automata with positive transitions:

► **Definition 3.1.** A register automaton is *positive* if for each transition  $c \xrightarrow{\varphi} c'$  the formula  $\varphi$  is positive:  $\varphi = \text{true}$  or  $\varphi$  uses the boolean operations  $\vee$  and  $\wedge$  only.

► **Theorem 3.2.** A data language is positive and NOFA-recognizable iff it is accepted by some positive register automaton.

Here, the approach is to regard a configuration of a positive register automaton as a state of a NOFRA. Conversely, an abstract transition  $j \xrightarrow{E} j'$  of a NOFA can be transformed into a transition  $j \xrightarrow{\varphi} j'$  of a register automaton for the conjunction  $\varphi$  of all equations in  $E$ , identifying  $k = \bullet, \bullet = k, k = \bar{k}$  with  $(k, \text{before}) = \bullet, \bullet = (k, \text{after}), (k, \text{before}) = (\bar{k}, \text{after})$ . A tweak of the initial states accounts for the requirement that registers are initially empty.

► **Remark 3.3.** Just like register automata are equivalent to finite-memory automata, positive register automata correspond to a restricted version of finite-memory automata called *finite-state unification-based automata (FSUBA)* [18, 32]. The original definition of the latter involves a fixed initial register assignment, which enables acceptance of non-positive languages. However, FSUBA with empty initial registers are equivalent to positive register automata; see the full paper for details. This implies in particular that positive register automata admit a decidable inclusion problem, in contrast to the case of unrestricted register automata [25]. Indeed, FSUBA translate into a more general model called *RNNA* [29, Sec. 6], for which inclusion is decidable. Tal [32] has given a direct decidability proof for FSUBA.

## 4 Monadic Second-Order Logic with Positive Equality Tests

As motivated in the Introduction, positive data languages are considered as expressing properties of data words involving positive statements about equality of data values. In the following we make this idea precise. For this purpose, we employ monadic second-order logic with equality tests, abbreviated  $\text{MSO}^\sim$  [4, 9, 25]. Its formulae are given by the grammar

$$\varphi, \psi := x < y \mid x \sim y \mid X(x) \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \exists x. \varphi \mid \exists X. \varphi \mid \forall x. \varphi \mid \forall X. \varphi,$$

where  $x, y$  range over first-order variables and  $X$  over monadic second-order variables. A formula is interpreted over a fixed data word  $w = a_1 \dots a_n \in \mathbb{A}^*$ . First-order variables represent positions, i.e. elements of the set  $\{1, \dots, n\}$ , and second-order variables represent subsets of  $\{1, \dots, n\}$ . The atomic formula  $x < y$  means “position  $x$  comes before position  $y$ ”, and  $x \sim y$  means “the same data value occurs at positions  $x$  and  $y$ ”. The interpretation of the remaining constructs is standard. A *sentence* is a formula without free variables. We write  $L(\varphi) \subseteq \mathbb{A}^*$  for the set of data words satisfying the sentence  $\varphi$ . For example, the languages  $L_0$  and  $L_1$  from the Introduction are defined by  $\varphi_0 = \forall y. \text{last}(y) \Rightarrow (\forall x. x < y \Rightarrow \neg(x \sim y))$ , where  $\text{last}(y) = \neg\exists z. y < z$  and  $\psi \Rightarrow \xi = \neg\psi \vee \xi$ , and by  $\varphi_1 = \exists x. \exists y. x < y \wedge x \sim y$ .

Recall that by standard rules of negation, every formula is equivalent to one in *negation normal form (NNF)*, where for each subformula  $\neg\varphi$  the formula  $\varphi$  is atomic.

► **Definition 4.1.** An  $\text{MSO}^\sim$  formula lies in  $\text{MSO}^{\sim,+}$  (*monadic second-order logic with positive equality tests*) if it admits an NNF containing no subformula of the form  $\neg(x \sim y)$ . A data language is  *$\text{MSO}^{\sim,+}$ -definable* if it is of the form  $L(\varphi)$  for an  $\text{MSO}^{\sim,+}$  sentence  $\varphi$ .

The above sentence  $\varphi_1$  lies in  $\text{MSO}^{\sim,+}$  but  $\varphi_0$  does not. The following is immediate:

► **Proposition 4.2.** *Every  $\text{MSO}^{\sim,+}$ -definable language is positive.*

► **Remark 4.3.** The logic  $\text{MSO}^\sim$  is more expressive than NOFAs [25], and the same holds for  $\text{MSO}^{\sim,+}$ : the language defined by the  $\text{MSO}^{\sim,+}$  sentence  $\varphi = \forall x. \exists y. (x < y \vee y < x) \wedge x \sim y$  (“no data value occurs only once”) is not NOFA-recognizable. However, within the class of NOFA-recognizable languages, positive and  $\text{MSO}^{\sim,+}$ -definable languages coincide:

► **Theorem 4.4.** *A NOFA-recognizable language is positive iff it is  $\text{MSO}^{\sim,+}$ -definable.*

Indeed, one can express the abstract acceptance condition of Proposition 2.15 in  $\text{MSO}^{\sim,+}$ .

## 5 Toposes for Names

In the remainder, we investigate positive data languages and their automata from a more conceptual perspective. Some familiarity with basic category theory (functors, natural transformations, (co-)limits, adjunctions) is required; see Mac Lane [22] for a gentle introduction.

Nominal sets and nominal renaming sets (Section 2.1) were initially introduced as a convenient abstract framework for reasoning about names, and related issues such as freshness, binding, and substitution. An alternative, and more general, approach uses the presheaf categories  $\mathbf{Set}^{\mathbb{A}}$  [31] and  $\mathbf{Set}^{\mathbb{A}^{\text{fin}}}$  [10]. The intuition behind each of these categories  $\mathcal{C}$  is very similar: one thinks of  $X \in \mathcal{C}$  as a collection of finitely supported objects, equipped with a renaming operation that extends renamings  $\rho: \mathbb{A} \rightarrow \mathbb{A}$  to the level of elements of  $X$ . The difference between the four categories lies in whether elements admit a *least* support, or just some finite support, and in whether renamings  $\rho$  are injective or arbitrary maps; see Figure 2. The last column classifies the respective finitely presentable objects, which underly automata. We now recall the latter concept and describe the categories in more detail.

**Finitely presentable objects.** A diagram  $D: I \rightarrow \mathcal{C}$  in a category  $\mathcal{C}$  is *directed* if its scheme  $I$  is a directed poset: every finite subset of  $I$  has an upper bound. A *directed colimit* is a colimit of a directed diagram. An object  $X$  of  $\mathcal{C}$  is *finitely presentable* if its hom-functor  $\mathcal{C}(X, -): \mathcal{C} \rightarrow \mathbf{Set}$  to the category of sets and functions preserves directed colimits. In many categories, finitely presentable objects correspond to the objects with a finite description. For example, the finitely presentable objects of  $\mathbf{Set}$  are precisely finite sets, and if  $\mathcal{C}$  is a variety of algebras (e.g. monoids, groups, rings), an algebra is a finitely presentable object of  $\mathcal{C}$  iff it is presentable by finitely many generators and relations [2, Thm. 3.12].

**Nominal (renaming) sets.** We let  $\mathbf{Nom}$  denote the category of nominal sets and  $\text{Perm}(\mathbb{A})$ -equivariant maps, and  $\mathbf{RnNom}$  the category of nominal renaming sets and  $\text{Fin}(\mathbb{A})$ -equivariant maps. Both categories are toposes, that is, they are finitely complete (with limits formed as in  $\mathbf{Set}$ ), cartesian closed, and admit a subobject classifier. Note that  $\mathbf{Nom}$  is a boolean topos (its subobject classifier is  $2 = \{0, 1\}$  with the trivial group action), which is not true for  $\mathbf{RnNom}$  [12, Sec. 5]. The next proposition provides a categorical description of orbit-finite nominal (renaming) sets; for nominal sets this result is well-known, see [26, Prop. 2.3.7] or [27, Thm. 5.16], and the statement for nominal renaming sets may be deduced from it.

Category	Objects	Least supp.	Renamings	Finitely pres. objects
<b>Nom</b>	nominal sets	yes	injective	orbit-finite sets
<b>RnNom</b>	nominal renaming sets	yes	arbitrary	orbit-finite sets
<b>Set<sup>ℐ</sup></b>	presheaves over ℐ	no	injective	super-finitary presheaves
<b>Set<sup>ℱ</sup></b>	presheaves over ℱ	no	arbitrary	super-finitary presheaves

■ **Figure 2** Toposes that model sets of finitely supported objects.

► **Proposition 5.1.** *A nominal (renaming) set is orbit-finite iff it is a finitely presentable object of **Nom** or **RnNom**, respectively.*

The forgetful functor  $U: \mathbf{RnNom} \rightarrow \mathbf{Nom}$  given by restricting the  $\text{Fin}(\mathbb{A})$ - to a  $\text{Perm}(\mathbb{A})$ -action has a left adjoint  $F: \mathbf{Nom} \rightarrow \mathbf{RnNom}$  [24, Thm. 2.6]. We refer to *op. cit.* for its explicit description, but remark that  $F(\mathbb{A}^{\#n}) = \mathbb{A}^n$  for every  $n \in \mathbb{N}$  [24, Thm. 3.7].

**Presheaves.** A (covariant) presheaf over a small category  $\mathcal{C}$  is a functor  $P: \mathcal{C} \rightarrow \mathbf{Set}$ . We write  $\mathbf{Set}^{\mathcal{C}}$  for the category of presheaves and natural transformations. We specifically consider presheaves over  $\mathbb{F}$  and  $\mathbb{I}$ , the categories whose objects are finite subsets  $S \subseteq_f \mathbb{A}$  and whose morphisms  $\rho: S \rightarrow T$  are functions or injective functions, respectively. The categories **Nom** and **RnNom** form full reflective subcategories of  $\mathbf{Set}^{\mathbb{I}}$  and  $\mathbf{Set}^{\mathbb{F}}$  via embeddings

$$I_*: \mathbf{Nom} \hookrightarrow \mathbf{Set}^{\mathbb{I}} \quad \text{and} \quad J_*: \mathbf{RnNom} \hookrightarrow \mathbf{Set}^{\mathbb{F}}.$$

Here,  $I_*$  is given for  $X \in \mathbf{Nom}$ ,  $S \subseteq_f \mathbb{A}$ ,  $\rho: S \rightarrow T$  in  $\mathbb{I}$  and  $f: X \rightarrow Y$  in **Nom** by

$$(I_*X)S = \{x \in X : \text{supp } x \subseteq S\}, \quad (I_*X)\rho(x) = \bar{\rho} \cdot x, \quad (I_*f)_S(x) = f(x),$$

where  $\bar{\rho} \in \text{Perm}(\mathbb{A})$  is any permutation extending the injective map  $\rho$ . The embedding  $J_*$  is defined analogously. In both cases, the essential image of the embedding consists precisely of the presheaves preserving pullbacks of injective maps, see [27, Thm. 6.8] and [12, Thm. 38]. Informally, a presheaf  $P \in \mathbf{Set}^{\mathcal{C}}$ , where  $\mathcal{C} \in \{\mathbb{I}, \mathbb{F}\}$ , specifies a set  $PS$  of  $S$ -supported objects for every  $S \subseteq_f \mathbb{A}$ , and the pullback preservation property asserts precisely that every object admits a least support. A presheaf  $P \in \mathbf{Set}^{\mathcal{C}}$  is *super-finitary* if there exists  $S \subseteq_f \mathbb{A}$  such that (i)  $PS'$  is a finite set for all  $S' \subseteq S$ , and (ii) for every  $T \subseteq_f \mathbb{A}$  and  $x \in PT$ , there exists  $S' \subseteq S$  and  $\rho \in \mathcal{C}(S', T)$  such that  $x \in P\rho[PS']$ . (This implies that  $PT$  is finite.) Such an  $S$  is called a *generating set* for  $P$ . The next proposition shows that super-finitary presheaves are the analogue of orbit-finite sets; see [1, Cor. 3.34] for the case  $\mathcal{C} = \mathbb{F}$ :

► **Proposition 5.2.** *For  $\mathcal{C} \in \{\mathbb{I}, \mathbb{F}\}$  and  $P \in \mathbf{Set}^{\mathcal{C}}$ , the following are equivalent: (i)  $P$  is super-finitary; (ii)  $P$  is finitely presentable; (iii) there exists an epimorphism (a componentwise surjective natural transformation)  $\coprod_{i \in I} \mathcal{C}(S_i, -) \twoheadrightarrow P$  with  $I$  finite and  $S_i \subseteq_f \mathbb{A}$ . Moreover, super-finitary presheaves are closed under sub-presheaves and finite products.*

To relate the two presheaf categories  $\mathbf{Set}^{\mathbb{I}}$  and  $\mathbf{Set}^{\mathbb{F}}$ , recall that every functor  $E: \mathcal{C} \rightarrow \mathcal{D}$  between small categories induces an adjunction (5.1), where the right adjoint  $E^*$  is given by  $E^*(P) = P \circ E$ , and the left adjoint sends a presheaf  $P \in \mathbf{Set}^{\mathcal{C}}$  to its *left Kan extension*  $\text{Lan}_E P$ . For the inclusion functor  $E: \mathbb{I} \hookrightarrow \mathbb{F}$ , we obtain the commutative diagram (5.2) of adjunctions. Here,  $I^*$  and  $J^*$  are the reflectors, i.e. the left adjoints of  $I_*$  and  $J_*$ .

$$\begin{array}{ccc}
 \mathbf{Set}^{\mathcal{C}} & \xleftarrow[\text{Lan}_E]{E^*} & \mathbf{Set}^{\mathcal{D}} \\
 & & (5.1)
 \end{array}
 \qquad
 \begin{array}{ccc}
 \mathbf{Set}^{\mathbb{I}} & \xleftarrow[\top]{E^*} & \mathbf{Set}^{\mathbb{F}} \\
 I_* \uparrow \vdash I^* & \xrightarrow[\text{Lan}_E]{} & J_* \downarrow \dashv J_* \\
 \mathbf{Nom} & \xleftarrow[U]{F} & \mathbf{RnNom}
 \end{array}
 \qquad
 (5.2)$$

► **Proposition 5.3.** *All functors in (5.2) preserve finitely presentable objects.*

Hence, the adjunctions (5.2) restrict to the full subcategories of finitely presentable objects.

## 6 Nondeterministic Automata in a Category

Our aim is to investigate nondeterministic automata and their languages in the toposes of Figure 2, and to compare their expressive power. To this end, we first introduce the required automata-theoretic concepts uniformly at the level of abstract categories.

► **Assumptions 6.1.** Fix a category  $\mathcal{C}$  with finite limits and (strong epi, mono)-factorizations. We assume that strong epimorphisms are stable under finite products (that is,  $e \times e'$  is a strong epimorphism whenever  $e$  and  $e'$  are) and pullbacks (that is, in every pullback square  $e \circ \bar{f} = f \circ \bar{e}$ , the morphism  $\bar{e}$  is a strong epimorphism whenever  $e$  is).

The (strong epi, mono)-factorization  $f = (A \xrightarrow{e} I \xrightarrow{m} B)$  of a morphism  $f: A \rightarrow B$  in  $\mathcal{C}$  is its *image factorization*, and the subobject represented by  $m$  is the *image* of  $f$ .

► **Example 6.2.** Every topos satisfies Assumptions 6.1, including **Set**, **Nom**, **RnNom**, **Set<sup>I</sup>** and **Set<sup>F</sup>**. Note that in a topos all epimorphisms are strong. In the five categories above, epi- and monomorphisms are the (componentwise) surjective and injective morphisms, resp. Pullbacks and finite products are formed (componentwise) at the level of underlying sets.

► **Definition 6.3.** A *language* over  $\Sigma \in \mathcal{C}$  is a family of subobjects of  $\Sigma^n$  for each  $n \in \mathbb{N}$ :

$$L = (m_n^{(L)}: L^{(n)} \rightarrow \Sigma^n)_{n \in \mathbb{N}}.$$

We write  $L \leq L'$  iff  $L^{(n)} \leq L'^{(n)}$  for all  $n$ , using the partial order  $\leq$  on subobjects of  $\Sigma^n$ .

► **Remark 6.4.** If  $\mathcal{C}$  is countably extensive (e.g. a topos with countable coproducts), languages correspond bijectively to subobjects of  $\Sigma^* = \coprod_{n \in \mathbb{N}} \Sigma^n$ . Indeed, every language  $L$  yields the subobject  $\coprod_n m_n^{(L)}: \coprod_n L^{(n)} \rightarrow \Sigma^*$ , and conversely every subobject of  $\Sigma^*$  is of this form. In particular, this holds in the categories of Example 6.2.

► **Definition 6.5.** A *nondeterministic  $\mathcal{C}$ -automaton* is a quintuple  $A = (Q, \Sigma, \delta, I, F)$  consisting of an object  $Q \in \mathcal{C}$  of *states*, an *input alphabet*  $\Sigma \in \mathcal{C}$ , and subobjects

$$m_\delta: \delta \rightarrow Q \times \Sigma \times Q, \quad m_I: I \rightarrow Q, \quad m_F: F \rightarrow Q,$$

representing *transitions*, *initial states*, and *final states*, respectively. A *morphism*  $h: A' \rightarrow A$  of nondeterministic  $\mathcal{C}$ -automata is given by a pair of morphisms  $h_s: Q' \rightarrow Q$  and  $h_a: \Sigma' \rightarrow \Sigma$  of  $\mathcal{C}$  that restrict as shown below (note that  $h_t$ ,  $h_i$  and  $h_f$  are uniquely determined):

$$\begin{array}{ccccc} \delta' & \xrightarrow{\quad h_t \quad} & \delta & & I' & \xrightarrow{\quad h_i \quad} & I & & F' & \xrightarrow{\quad h_f \quad} & F \\ m_{\delta'} \downarrow & & \downarrow m_\delta & & m_{I'} \downarrow & & \downarrow m_I & & m_{F'} \downarrow & & \downarrow m_F \\ Q' \times \Sigma' \times Q' & \xrightarrow{\quad h_s \times h_a \times h_s \quad} & Q \times \Sigma \times Q & & Q' & \xrightarrow{\quad h_s \quad} & Q & & Q' & \xrightarrow{\quad h_s \quad} & Q \end{array} \quad (6.1)$$

We write  $\mathbf{NAut}(\mathcal{C})$  for the category of nondeterministic automata in  $\mathcal{C}$  and their morphisms, and  $\mathbf{NAut}_{\text{fp}}(\mathcal{C})$  for its full subcategory given by *nondeterministic fp-automata*, viz. automata where  $Q, \Sigma, \delta, I, F$  are finitely presentable objects of  $\mathcal{C}$ .

► **Definition 6.6.** For every nondeterministic  $\mathcal{C}$ -automaton  $A = (Q, \Sigma, \delta, I, F)$ , its *accepted language* is the language  $L(A)$  over  $\Sigma$  given as follows:

1.  $m_{L(A)}^{(0)}: L^{(0)}(A) \rightarrow 1 = \Sigma^0$  is the image of the unique morphism  $I \cap F \xrightarrow{!} 1$ , where  $1$  is the terminal object of  $\mathcal{C}$  and  $I \cap F$  is the intersection (pullback) of  $m_I$  and  $m_F$ .

2. For  $n > 0$ , the subobject  $m_{L(A)}^{(n)}: L^{(n)}(A) \rightarrow \Sigma^n$  is defined via the commutative diagram

$$\begin{array}{ccccc} L^{(n)}(A) & \xleftarrow{e_{n,A}} & \text{AccRun}_A^{(n)} & \xrightarrow{\bar{d}_{n,A}} & \delta^n \\ m_{L(A)}^{(n)} \downarrow & & \downarrow \bar{m}_\delta^{(n)} & & \downarrow m_\delta^n \\ \Sigma^n & \xleftarrow{p_{n,A}} & I \times (\Sigma \times Q)^{n-1} \times \Sigma \times F & \xrightarrow{d_{n,A}} & (Q \times \Sigma \times Q)^n \end{array}$$

Here, letting  $\Delta: Q \rightarrow Q \times Q$  denote the diagonal,  $d_{n,A}$  is the monomorphism

$$I \times (\Sigma \times Q)^{n-1} \times \Sigma \times F \xrightarrow{m_I \times (\text{id} \times \Delta)^{n-1} \times \text{id} \times m_F} Q \times (\Sigma \times Q \times Q)^{n-1} \times \Sigma \times Q \cong (Q \times \Sigma \times Q)^n,$$

the morphisms  $\bar{d}_{n,A}$  and  $\bar{m}_\delta^{(n)}$  form the pullback of  $d_{n,A}$  and  $m_\delta^n$ , the morphism  $p_{n,A}$  is the projection, and  $e_{n,A}$  and  $m_{L(A)}^{(n)}$  form the image factorization of  $p_{n,A} \circ \bar{m}_\delta^{(n)}$ .

► **Example 6.7.**

1. A nondeterministic fp-automaton in **Set** is a classical nondeterministic finite automaton. The pullback  $\text{AccRun}_A^{(n)}$  is the set of accepting runs of length  $n$ , hence  $L(A)$  is the usual accepted language: the set of words with an accepting run.
2. A nondeterministic fp-automaton in **Nom** or **RnNom** with alphabet  $\Sigma = \mathbb{A}$  is a NOFA or NOFRA, respectively. The two notions of accepted language in Definition 2.3 and Definition 6.6 match, that is,  $L(A)$  is the set of words with an accepting run.
3. In the next section, we will also look into nondeterministic **Set<sup>I</sup>**- and **Set<sup>F</sup>**-automata.

► **Remark 6.8.** Readers familiar with coalgebras [28] may note that if  $\mathcal{C}$  is a topos, the final states and transitions of a nondeterministic  $\mathcal{C}$ -automaton correspond to a coalgebra  $\gamma: Q \rightarrow \Omega \times (\mathcal{P}Q)^\Sigma$  where  $\Omega$  is the subobject classifier and  $\mathcal{P}: \mathcal{C} \rightarrow \mathcal{C}$  is the covariant power object functor [16, Sec. A.2.3]. We expect our above definition of accepted language to match the one given by coalgebraic trace semantics [15, 30], with the required arguments relying on the internal logic of the topos  $\mathcal{C}$ . Details are left for future work; we have found that the present relational approach to automata leads to shorter and more direct proofs.

► **Proposition 6.9.** *Let  $h: A' \rightarrow A$  be an  $\mathbf{NAut}(\mathcal{C})$ -morphism where  $\Sigma' = \Sigma$  and  $h_a = \text{id}_\Sigma$ .*

1. *The accepted language of  $A'$  is contained in that of  $A$ , that is,  $L(A') \leq L(A)$ .*
2. *If  $h_s$  is strongly epic in  $\mathcal{C}$  and the squares (6.1) are pullbacks, then  $L(A') = L(A)$ .*

Hence, the construction  $A \mapsto A'$  of Remark 2.6 indeed yields an equivalent NOFA.

► **Proposition 6.10.** *Let  $\mathcal{C}$  and  $\mathcal{D}$  be categories satisfying the Assumptions 6.1.*

1. *Every functor  $G: \mathcal{C} \rightarrow \mathcal{D}$  lifts to a functor  $\bar{G}: \mathbf{NAut}(\mathcal{C}) \rightarrow \mathbf{NAut}(\mathcal{D})$  defined by*

$$\bar{G}(Q, \Sigma, \delta, I, F) = (GQ, G\Sigma, \overline{G\delta}, \overline{GI}, \overline{GF}) \quad \text{and} \quad \bar{G}f = Gf.$$

*Here,  $\overline{G\delta}$ ,  $\overline{GI}$ ,  $\overline{GF}$  are the images of the morphisms shown below, with  $\text{can}$  denoting the canonical morphism induced by the product projections:*

$$G\delta \xrightarrow{Gm_\delta} G(Q \times \Sigma \times Q) \xrightarrow{\text{can}} GQ \times G\Sigma \times GQ, \quad GI \xrightarrow{Gm_I} GQ, \quad GF \xrightarrow{Gm_F} GQ.$$

2. *Every adjunction  $L \dashv R: \mathcal{C} \rightarrow \mathcal{D}$  lifts to an adjunction  $\bar{L} \dashv \bar{R}: \mathbf{NAut}(\mathcal{C}) \rightarrow \mathbf{NAut}(\mathcal{D})$ .*

In particular, the adjunctions (5.2) lift to adjunctions between the respective categories of nondeterministic automata, which in turn restrict to fp-automata by Proposition 5.3:

$$\begin{array}{ccc}
\mathbf{NAut}_{\text{fp}}(\mathbf{Set}^{\mathbb{I}}) & \xleftarrow[\perp]{\bar{E}^*} & \mathbf{NAut}_{\text{fp}}(\mathbf{Set}^{\mathbb{F}}) \\
\tilde{I}_* \uparrow \downarrow \tilde{I}^* & \xrightarrow[\text{Lan}_E]{\perp} & \tilde{J}^* \downarrow \uparrow \tilde{J}_* \\
\mathbf{NAut}_{\text{fp}}(\mathbf{Nom}) & \xleftarrow[\bar{U}]{\bar{F}} & \mathbf{NAut}_{\text{fp}}(\mathbf{RnNom})
\end{array} \tag{6.2}$$

The positive closure  $A \mapsto \bar{A}$  of Construction 2.7, which is key to our results in Sections 2 through 4, is an instance of the proposition since  $\bar{A} = \bar{F}A$  for the left adjoint  $F: \mathbf{Nom} \rightarrow \mathbf{RnNom}$ .

## 7 Nondeterministic Presheaf Automata

We proceed to relate the expressive power of the four automata models in (6.2). Specifically, for  $\mathcal{C} \in \{\mathbb{I}, \mathbb{F}\}$  we consider nondeterministic  $\mathbf{Set}^{\mathcal{C}}$ -automata  $A = (Q, \Sigma, \delta, I, F)$  with a super-finitary (= finitely presentable) presheaf  $Q$  of states and input alphabet  $\Sigma = V_{\mathcal{C}} \in \mathbf{Set}^{\mathcal{C}}$ , for the inclusion functor  $V_{\mathcal{C}}(S) = S$ . (This implies that  $\delta$ ,  $I$  and  $F$  are super-finitary by Proposition 5.2.) Note that  $V_{\mathcal{C}}$  corresponds to the input alphabet  $\mathbb{A}$  used for NOF(R)As:

$$V_{\mathbb{I}} = I_*(\mathbb{A}) \quad \text{and} \quad V_{\mathbb{F}} = J_*(\mathbb{A}) = \text{Lan}_E(V_{\mathbb{I}}).$$

A language in  $\mathbf{Set}^{\mathcal{C}}$  is a sub-presheaf  $L \subseteq V_{\mathcal{C}}^*$ , or equivalently a family of sub-presheaves  $L^{(n)} \subseteq V_{\mathcal{C}}^n$  for  $n \in \mathbb{N}$  (Definition 6.3 and Remark 6.4). Here,  $V_{\mathcal{C}}^*(S) = S^*$ , the set of words over the finite alphabet  $S \subseteq_f \mathbb{A}$ , and  $V_{\mathcal{C}}^n(S) = S^n$ , the subset of words of length  $n$ .

► **Remark 7.1.** For the sake of distinction, we refer to languages in  $\mathbf{Set}^{\mathcal{C}}$  as *presheaf languages*, and to subsets of  $\mathbb{A}^*$  as *word languages*. Both concepts are closely related: Every presheaf language  $L \subseteq V_{\mathbb{I}}^*$  in  $\mathbf{Set}^{\mathbb{I}}$  induces a  $\text{Perm}(\mathbb{A})$ -equivariant word language  $W(L) \subseteq \mathbb{A}^*$  given by  $W(L) = \bigcup_{S \subseteq_f \mathbb{A}} L(S)$ , and, conversely, every  $\text{Perm}(\mathbb{A})$ -equivariant word language  $K \subseteq \mathbb{A}^*$  induces a presheaf language  $P(K) \subseteq V_{\mathbb{I}}^*$  given by  $[P(K)]S = K \cap S^*$  for  $S \subseteq_f \mathbb{A}$ . Analogously for presheaf languages in  $\mathbf{Set}^{\mathbb{F}}$  and  $\text{Fin}(\mathbb{A})$ -equivariant word languages. In both cases, these translations almost yield a bijective correspondence: one has  $K = W(P(K))$ , but generally only  $L \subseteq P(W(L))$ . For instance, for  $L \subseteq V_{\mathbb{F}}^*$  given by  $L(\emptyset) = \emptyset$  and  $L(S) = \{\varepsilon\}$  for  $S \neq \emptyset$  one has  $[P(W(L))]\emptyset = \{\varepsilon\}$ , so  $L \subsetneq P(W(L))$ . The equality  $L = P(W(L))$  holds iff  $L$  is *downwards closed*, that is,  $L(S') = L(S) \cap (S')^*$  for all  $S' \subseteq S \subseteq_f \mathbb{A}$ .

The presheaf version of positive word languages and positive closures is as follows:

► **Definition 7.2.** Let  $L \subseteq V_{\mathbb{I}}^*$  be a presheaf language in  $\mathbf{Set}^{\mathbb{I}}$ .

1. The language  $L$  is *positive* if  $L = KE$  for some (unique) language  $K \subseteq V_{\mathbb{F}}^*$  in  $\mathbf{Set}^{\mathbb{F}}$ .
2. A *positive closure* of  $L$  is a language  $\bar{L}$  in  $\mathbf{Set}^{\mathbb{F}}$  such that  $L \subseteq \bar{L}E$  and  $\bar{L}$  is minimal with that property, that is,  $\bar{L} \subseteq K$  for every language  $K \subseteq V_{\mathbb{F}}^*$  in  $\mathbf{Set}^{\mathbb{F}}$  such that  $L \subseteq KE$ .

A positive closure is clearly unique; its existence is ensured by the next proposition, which is proved using the universal property of left Kan extensions.

► **Proposition 7.3.** *The positive closure of  $L \subseteq V_{\mathbb{I}}^*$  is given by the image of the morphism*

$$\varphi: \text{Lan}_E(L) \xrightarrow{\text{Lan}_E(\iota)} \text{Lan}_E(V_{\mathbb{I}}^*) \cong \coprod_k \text{Lan}_E(V_{\mathbb{I}}^k) \xrightarrow{\coprod_k \text{can}_k} \coprod_k \text{Lan}_E(V_{\mathbb{I}})^k = \coprod_k V_{\mathbb{F}}^k = V_{\mathbb{F}}^*$$

where  $\iota: L \hookrightarrow V_{\mathbb{I}}^*$  is the inclusion, the isomorphism witnesses preservation of coproducts by the left adjoint  $\text{Lan}_E$ , and  $\text{can}_k$  is the canonical map induced by the product projections.

► **Remark 7.4.** A presheaf  $P \in \mathbf{Set}^{\mathbb{I}}$  is *strong* if  $P = I_*(X)$  for a strong nominal set  $X$ . Since  $I_*$  preserves coproducts, (super-finitary) strong presheaves are exactly (finite) coproducts  $\coprod_{j \in J} \mathbb{I}(S_j, -)$  of representable presheaves. By Proposition 5.2 and Proposition 6.9, every super-finitary  $\mathbf{Set}^{\mathbb{I}}$ -automaton is equivalent to one whose presheaf of states is strong. Given such an automaton  $A$  with states  $Q = \coprod_{j \in J} \mathbb{I}(S_j, -)$ , applying the lifted left adjoint  $\overline{\mathbf{Lan}}_E$  yields a super-finitary  $\mathbf{Set}^{\mathbb{F}}$ -automaton  $\bar{A}$  with states  $\mathbf{Lan}_E(Q) = \coprod_{j \in J} \mathbb{F}(S_j, -)$ , using that  $\mathbf{Lan}_E$  preserves coproducts and representables (see e.g. [22, Ex. X.3.2]). This is the analogue of Construction 2.7 for presheaf automata. Similar to Proposition 2.8, we have

► **Proposition 7.5.** *For every super-finitary nondeterministic  $\mathbf{Set}^{\mathbb{I}}$ -automaton  $A$  with a strong presheaf of states, the  $\mathbf{Set}^{\mathbb{F}}$ -automaton  $\bar{A} = \overline{\mathbf{Lan}}_E(A)$  accepts the language  $\overline{L(A)}$ .*

While by definition nondeterministic presheaf automata accept presheaf languages, using Remark 7.1 we can also naturally associate a word language semantics to them:

► **Definition 7.6.**

1. The word language *accepted* by a nondeterministic  $\mathbf{Set}^{\mathcal{C}}$ -automaton  $A$  is  $W(L(A)) \subseteq \mathbb{A}^*$ , the word language induced by the presheaf language of  $A$ .
2. A word language  $L \subseteq \mathbb{A}^*$  is  *$\mathbf{Set}^{\mathcal{C}}$ -recognizable* if there exists a super-finitary nondeterministic  $\mathbf{Set}^{\mathcal{C}}$ -automaton accepting it.

This enables a classification of the expressive power of nondeterministic  $\mathbf{Set}^{\mathcal{C}}$ -automata:

► **Theorem 7.7.**

1. *A word language is NOFA-recognizable iff it is  $\mathbf{Set}^{\mathbb{I}}$ -recognizable.*
2. *A word language is positive and NOFA-recognizable iff it is  $\mathbf{Set}^{\mathbb{F}}$ -recognizable.*

For item 1 one shows that the functors  $\bar{I}_*$  and  $\bar{I}^*$  of (6.2) preserve the accepted word languages of automata. For item 2 one uses Proposition 7.5 and the observation that every nondeterministic  $\mathbf{Set}^{\mathbb{F}}$ -automaton accepts a positive word language.

This shows that the theory of data languages can be based on presheaves rather than nominal sets [6]. In particular, the conceptual difference between the two approaches (viz. existence of least supports) is largely inessential from the perspective of automata theory.

## 8 Conclusions and Future Work

We have characterized positive data languages recognizable by NOFAs in terms of register automata, logic, and category theory; see Figure 1 for a summary of our contributions. Our results underline the phenomenon that weak classes of data languages tend to have a rich theory and admit many equivalent perspectives, paralleling classical regular languages over finite alphabets. For example, a similar observation has been made for data languages recognizable by orbit-finite nominal monoids [4, 8, 9].

The logic  $\text{MSO}^{\sim,+}$  defines positive data languages, but is more expressive than NOFAs. Identifying a suitable syntactic fragment of  $\text{MSO}^{\sim,+}$  that captures precisely the positive NOFA-recognizable languages remains an open problem. The same holds for the decidability of the satisfiability problem for  $\text{MSO}^{\sim,+}$ , which is known to be undecidable for  $\text{MSO}^{\sim}$  [20]. On a related note, it might be interesting to characterize the expressive power of full  $\text{MSO}^{\sim,+}$ . Specifically, does it capture precisely the  $\text{MSO}^{\sim}$ -definable positive languages?

Finally, besides register automata, a number of further automata models for data languages have been proposed, most notably pebble automata [25] and data automata [5, 7]. In general, these models differ in their expressive power. However, it is conceivable that some or all of them may become equivalent when restricted to positive data languages.

## References

- 1 Jiří Adámek, Stefan Milius, Lurdes Sousa, and Thorsten Wißmann. On finitary functors. *Theory Appl. Categ.*, 34(37):1134–1164, 2019.
- 2 Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1994.
- 3 Michał Bielecki, Jan Hidders, Jan Paredaens, Jerzy Tyszkiewicz, and Jan Van den Bussche. Navigating with a browser. In *Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, volume 2380 of *Lect. Notes Comput. Sci.*, pages 764–775. Springer, 2002.
- 4 Mikołaj Bojańczyk. Nominal monoids. *Theory Comput. Syst.*, 53(2):194–222, 2013. doi:10.1007/s00224-013-9464-1.
- 5 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. In *Proc. 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 10–19. ACM, 2006.
- 6 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.
- 7 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *Proc. 21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 7–16. IEEE Computer Society, 2006.
- 8 Mikołaj Bojańczyk and Rafał Stefański. Single-use automata and transducers for infinite alphabets. In *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPICs*, pages 113:1–113:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.113.
- 9 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. Logics with rigidly guarded data tests. *Log. Methods Comput. Sci.*, 11(3), 2015.
- 10 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*, pages 193–202. IEEE Computer Society, 1999.
- 11 Murdoch J. Gabbay. Nominal renaming sets (technical report), 2007. URL: <http://gabbay.org.uk/papers/nomrs-tr.pdf>.
- 12 Murdoch J. Gabbay and Martin Hofmann. Nominal renaming sets. In *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2008)*, pages 158–173. Springer, 2008.
- 13 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*, pages 214–224. IEEE Computer Society, 1999.
- 14 Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *High. Order Symb. Comput.*, 19(2-3):283–304, 2006.
- 15 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Log. Methods Comput. Sci.*, 3(4:11):1–36, 2007.
- 16 Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Oxford Logic Guides. Oxford Univ. Press, 2002.
- 17 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 18 Michael Kaminski and Tony Tan. Regular expressions for languages over infinite alphabets. *Fundam. Informaticae*, 69(3):301–318, 2006.
- 19 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010.
- 20 Bartek Klin, Sławomir Lasota, and Szymon Torunczyk. Nondeterministic and conondeterministic implies deterministic, for data languages. In *Proc. 24th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2021)*, volume 12650 of *Lect. Notes Comput. Sci.*, pages 365–384. Springer, 2021.

- 21 Klaas Kürtz, Ralf Küsters, and Thomas Wilke. Selecting theories and nonce generation for recursive protocols. In *Proc. 2007 ACM Workshop on Formal Methods in Security Engineering (FMSE 2007)*, pages 61–70. ACM, 2007.
- 22 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.
- 23 Stefan Milius and Henning Urbat. Equational axiomatization of algebras with structure. In *Proc. 22nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2019)*, volume 11425 of *Lect. Notes Comput. Sci.*, pages 400–417. Springer, 2019. doi:10.1007/978-3-030-17127-8\_23.
- 24 Joshua Moerman and Jurriaan Rot. Separation and Renaming in Nominal Sets. In *Proc. 28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.
- 25 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
- 26 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. PhD thesis, University of Leicester, 2012.
- 27 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.
- 28 Jan J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
- 29 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *Proc. 20th International Conference on Foundations of Software Science and Computation Structures, (FOSSACS 2017)*, volume 10203 of *Lect. Notes Comput. Sci.*, pages 124–142, 2017.
- 30 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Log. Methods Comput. Sci.*, 9(1:9), 2013.
- 31 Ian Stark. Categorical models for local names. *LISP Symb. Comput.*, 9(1):77–107, 1996.
- 32 A. Tal. Decidability of inclusion for unification based automata. Master’s thesis, Department of Computer Science, Technion – Israel Institute of Technology, 1999.



# Parameterized Analysis of the Cops and Robber Game

Harmender Gahlawat   

Ben-Gurion University of the Negev, Beersheba, Israel

Meirav Zehavi   

Ben-Gurion University of the Negev, Beersheba, Israel

---

## Abstract

*Pursuit-evasion games* have been intensively studied for several decades due to their numerous applications in artificial intelligence, robot motion planning, database theory, distributed computing, and algorithmic theory. COPS AND ROBBER (CNR) is one of the most well-known pursuit-evasion games played on graphs, where multiple *cops* pursue a single *robber*. The aim is to compute the *cop number* of a graph,  $k$ , which is the minimum number of cops that ensures the *capture* of the robber.

From the viewpoint of parameterized complexity, CNR is W[2]-hard parameterized by  $k$  [Fomin et al., TCS, 2010]. Thus, we study structural parameters of the input graph. We begin with the *vertex cover number* (vcn). First, we establish that  $k \leq \frac{vcn}{3} + 1$ . Second, we prove that CNR parameterized by vcn is FPT by designing an exponential kernel. We complement this result by showing that it is unlikely for CNR parameterized by vcn to admit a polynomial compression. We extend our exponential kernels to the parameters *cluster vertex deletion number* and *deletion to stars number*, and design a linear vertex kernel for *neighborhood diversity*. Additionally, we extend all of our results to several well-studied variations of CNR.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms; Mathematics of computing  $\rightarrow$  Graph algorithms

**Keywords and phrases** Cops and Robber, Kernelization, Graph Searching, Fixed parameter tractability

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.49

**Related Version** *Full Version*: <https://doi.org/10.48550/arXiv.2307.04594> [29]

**Funding** This research was supported by the European Research Council (ERC) grant titled PARAPATH.

## 1 Introduction

In *pursuit-evasion*, a set of agents, called *pursuers*, plan to catch one or multiple *evaders*. Classically, pursuit-evasion games were played on geometric setups, where pursuers and evaders move on the plane [35, 49]. Parsons [48] formulated pursuit-evasion on graphs to model the search for a person trapped in caves, giving rise to the field of graph searching. Since then, pursuit-evasion has been studied extensively, having applications in artificial intelligence [36], robot motion planning [16, 40], constraint satisfaction and database theory [31, 32, 33], distributed computing [4, 18] and network decontamination [45], and significant implications in graph theory and algorithms [1, 25, 30, 54].

COPS AND ROBBER (CNR) is one of the most intensively studied pursuit-evasion games on graphs, where a set of cops pursue a single robber. Players move in discrete time steps alternately, starting with the cops. In each move, a player can move to an adjacent vertex, and the cops win by *capturing* the robber (i.e., if a cop and the robber occupy the same vertex). The goal is to compute the *cop number* of a graph  $G$ , denoted  $c(G)$ , which is the minimum number of cops required to win in  $G$ . We define the game formally in Section 2. CNR is well studied in the artificial intelligence literature under the name MOVING TARGET



© Harmender Gahlawat and Meirav Zehavi;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 49; pp. 49:1–49:17

Leibniz International Proceedings in Informatics



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

PURSUIT (MTP) [37], where we consider sub-optimal but faster strategies from an applicative point of view. The results have found numerous applications in game design, police chasing, path planning, and robot motion planning [5, 44, 56].

Determining the parameterized complexity of games is a well-studied research topic [10, 11, 52]. Most pursuit-evasion games are AW[\*]-hard [53]. In particular, CNR is W[2]-hard parameterized by  $c(G)$  [27]. Thus, we consider structural parameterizations, focusing on *kernelization*, polynomial-time preprocessing with a parametric guarantee. Due to its profound impact, kernelization was termed “the lost continent of polynomial time” [26]. We begin with the most studied structural parameter: the *vertex cover number* ( $\text{vcn}$ ) of the input graph. We bound  $c(G)$  in terms of  $\text{vcn}$ , as well as achieve both positive and negative results concerning the kernelization complexity of CNR parameterized by  $\text{vcn}$ . We generalize our kernelization results to the smaller parameters *cluster vertex deletion number* ( $\text{cvd}$ ) and *deletion to stars number* ( $\text{dts}$ ), as well as to the parameter *neighborhood diversity* ( $\text{nd}$ ). Furthermore, we extend all our results to several well-studied variants of CNR.

The choice of  $\text{vcn}$  to study pursuit-evasion games is natural due to various scenarios where  $\text{vcn}$  is significantly smaller than the graph size. For example, this includes scenarios where we model the existence of one or few (possibly interconnected) central hubs – for illustration, suppose an intruder is hiding in a system of buildings where we have only few corridors but a large number of rooms, or suppose we have few virtual servers with many stations (e.g., of private users) that can communicate only with the servers. Furthermore,  $\text{vcn}$  is one of the most efficiently computable parameters from both approximation [55] and parameterized [17] points of view, making it fit from an applicative perspective even when a vertex cover is not given along with the input. Moreover,  $\text{vcn}$  is the best choice for proving negative results – indeed, our negative result on the kernelization complexity of CNR for  $\text{vcn}$  implies the same for many other well-known smaller parameters such as treewidth, treedepth and feedback vertex set [28]. One shortcoming of  $\text{vcn}$  as a parameter is that it is large for some simple (and easy to resolve) dense graphs like cliques. However, we generalize our kernel to  $\text{cvd}$ , which is small for these dense graphs, and to  $\text{dts}$ . Furthermore, we design a *linear* kernel for the well-studied parameter  $\text{nd}$ . We further discuss the utility of our kernels in the Conclusion.

**Brief Survey.** CNR was independently introduced by Quilliot [51] and Nowakowski and Winkler [46] with exactly one cop.<sup>1</sup> Aigner and Fromme [2] generalized CNR to multiple cops and defined the *cop number* of a graph. We refer to the book [9] for details.

The computational complexity of CNR is a challenging research subject. On the positive side, Berarducci and Intrigila [6] gave a backtracking algorithm to decide if  $G$  is  $k$ -copwin in  $\mathcal{O}(n^{2k+1})$  time. On the negative side, Fomin et al. [27] proved that CNR is NP-hard, and W[2]-hard parameterized by  $k$ . Moreover, Mamino [43] showed that CNR is PSPACE-hard, and later, Kinnersley [39] proved that CNR is, in fact, EXPTIME-complete. Recently, Brandt et al. [14] proved that, conditioned on (*Strong*) *Exponential Time Hypothesis*, the time complexity of any algorithm for CNR is  $(\Omega(n^{k-o(1)})) 2^{\Omega(\sqrt{n})}$ . Since CNR admits an XP-time algorithm, it is sensible to bound the cop number for various graph classes or by structural parameters. Nowadays, we know that the cop number is at most 3 for toroidal graphs [41], 9 for unit-disk graphs [7], 13 for string graphs [20], and bounded for bounded genus graphs [12] and minor-free graphs [3]. Moreover,  $c(G) \leq \frac{\text{tw}(G)}{2} + 1$  [38] and  $c(G) \leq \text{cw}(G)$  [27], where  $\text{tw}(G)$  and  $\text{cw}(G)$  are the treewidth and the cliquewidth of  $G$ , respectively.

---

<sup>1</sup> In fact, a specific instance of CNR was given as a puzzle in the book [24] already in 1917.

**Our Contribution.** We conduct a comprehensive analysis of CNR parameterized by  $\text{vcn}$ . Due to space constraints, the proofs of the claims marked by (\*) and the claims for which we only provide a proof sketch are deferred to the full version [29].

We start by bounding the cop number of a graph:

► **Theorem 1.** *For a graph  $G$ ,  $c(G) \leq \frac{\text{vcn}}{3} + 1$ .*

The proof is based on the application of three reduction rules. Each of our rules controls its own cop, which guards at least three vertices from the vertex cover. Once our rules are no longer applicable, we exhibit that the remaining unguarded part of the graph is of a special form, and the usage of only two additional cops suffices. We complement Theorem 1 with an argument that it might be difficult to improve this bound further using techniques similar to ours.

Second, we prove that CNR parameterized by  $\text{vcn}$  is FPT by designing a kernelization algorithm:

► **Theorem 2.** *CNR parameterized by  $\text{vcn}$  admits a kernel with at most  $\text{vcn} + \frac{2^{\text{vcn}}}{\sqrt{\text{vcn}}}$  vertices.*

Our kernel is also based on the application of reduction rules. However, these rules are very different than those used for the proof of Theorem 1. While our main rule is quite standard in kernelization (involving the removal of false twins), the proof of its correctness is (arguably) not. Theorem 2, Theorem 1, and the XP-algorithm (Proposition 12) yield the following corollary:

► **Corollary 3.** *CNR is FPT parameterized by  $\text{vcn}$ , and solvable in  $(\text{vcn} + \frac{2^{\text{vcn}}}{\sqrt{\text{vcn}}})^{\frac{\text{vcn}}{3} + 2} \cdot n^{\mathcal{O}(1)}$  time.*

We complement our kernel by showing that it is unlikely for CNR to admit polynomial compression, by providing a *polynomial parameter transformation* from RED-BLUE DOMINATING SET. Our reduction makes non-trivial use of a known construction of a special graph having high girth and high minimum degree.

► **Theorem 4.** *CNR parameterized by  $\text{vcn}$  does not admit polynomial compression, unless  $NP \subseteq \text{coNP}/\text{poly}$ .*

We also present a linear vertex kernel parameterized by  $\text{nd}$  for CNR, LAZY CNR, and COPS AND ATTACKING ROBBER, and a quadratic vertex kernel for COPS AND FAST ROBBER and FULLY ACTIVE CNR:

► **Theorem 5 (\*)**. *CNR, LAZY CNR, and COPS AND ATTACKING ROBBER parameterized by  $\text{nd}$  admits a kernel with at most  $\text{nd}$  vertices. Moreover, COPS AND FAST ROBBER and FULLY ACTIVE CNR parameterized by  $\text{nd}$  admit a kernel with at most  $\text{nd}^2$  vertices.*

On the positive side, we extend our exponential kernel to smaller parameters,  $\text{cvd}$  and  $\text{dts}$ :

► **Theorem 6.** *CNR parameterized by  $\text{cvd}$  admits a kernel with at most  $2^{2^{\text{cvd}} + \sqrt{\text{cvd}}}$  vertices. Moreover, CNR parameterized by  $\text{dts}$  admits a kernel with at most  $2^{2^{\text{dts}} + \text{dts}^{1.5}}$  vertices.*

Several variants of CNR have been studied due to their copious applications. We extend our results, parameterized by  $\text{vcn}$ , to some of the most well-studied ones. We define these variants (and used notations) in Section 2. We first bound the cop number of these variants by  $\text{vcn}$ :

► **Theorem 7** (\*). For a graph  $G$ : (1)  $c_{\text{lazy}} \leq \frac{\text{vcn}}{2} + 1$ ; (2)  $c_{\text{attack}} \leq \frac{\text{vcn}}{2} + 1$ ; (3)  $c_{\text{active}}(G) \leq \text{vcn}$ ; (4)  $c_{\text{surround}}(G) \leq \text{vcn}$ ; (5)  $c_s(G) \leq \text{vcn}$  (for any value of  $s$ ); (6) for a strongly connected orientation  $\vec{G}$  of  $G$ ,  $c(\vec{G}) \leq \text{vcn}$ .

We also extend our exponential kernel to these variants:

► **Theorem 8**. COPS AND ATTACKING ROBBER and LAZY CNR parameterized by  $\text{vcn}$  admit a kernel with at most  $\text{vcn} + \frac{2^{\text{vcn}}}{\sqrt{\text{vcn}}}$  vertices. Moreover, CNR on strongly connected directed graphs admits a kernel with at most  $3^{\text{vcn}} + \text{vcn}$  vertices.

Then, we present a slightly more general kernelization that works for most variants of the game:

► **Theorem 9**. FULLY ACTIVE CNR, COPS AND FAST ROBBER, and SURROUNDING CNR parameterized by  $\text{vcn}$  admit a kernel with at most  $\text{vcn} + \text{vcn} \cdot 2^{\text{vcn}}$  vertices.

We complement our exponential kernels for these variants by arguing about their compressibility:

► **Theorem 10** (\*). LAZY CNR, COPS AND ATTACKING ROBBER, COPS AND FAST ROBBER, FULLY ACTIVE CNR, and CNR on strongly connected directed and oriented graphs parameterized by  $\text{vcn}$  do not admit a polynomial compression, unless  $\text{NP} \subseteq \text{coNP}/\text{poly}$ .

## 2 Preliminaries

For  $\ell \in \mathbb{N}$ , let  $[\ell] = \{1, \dots, \ell\}$ . Whenever we mention  $\frac{a}{b}$ , we mean  $\lceil \frac{a}{b} \rceil$ .

**Graph Theory.** We only consider finite, connected<sup>2</sup>, and simple graphs. For a graph  $G$ , we denote  $|V(G)|$  by  $n$ . Let  $v \in V(G)$ . Then,  $N(v) = \{u \mid uv \in E(G)\}$  and  $N[v] = N(v) \cup \{v\}$ . For  $X \subseteq V(G)$ , let  $N_X(v) = N(v) \cap X$  and  $N_X[v] = N[v] \cap X$ . We say that  $v$  dominates  $u$  if  $u \in N[v]$ . The *girth* of  $G$  is the length of a shortest cycle contained in  $G$ . A  $u, v$ -path is a path with endpoints  $u$  and  $v$ . A path is *isometric* if it is a shortest path between its endpoints. For a directed graph  $\vec{G}$ , let  $N^+(u)$  and  $N^-(u)$  denote the set of out-neighbors and in-neighbors of  $u$ , respectively.

Let  $G$  be a graph and  $U \subseteq V(G)$ . Then,  $G[U]$  denotes the subgraph of  $G$  induced by  $U$ . A set  $U \subseteq V(G)$  is a *vertex cover* if  $G[V(G) \setminus U]$  is an independent set. The minimum cardinality of a vertex cover of  $G$  is its *vertex cover number* ( $\text{vcn}$ ). Moreover,  $U$  is a *cluster vertex deletion set* if  $G[V(G) \setminus U]$  is a disjoint union of cliques. The minimum size of a cluster vertex deletion set of a graph is its *cluster vertex deletion number* ( $\text{cvd}$ ). Additionally,  $U$  is a *deletion to stars set* if  $G[V(G) \setminus U]$  is a disjoint union of star graphs. The minimum size of a deletion to stars set of a graph is its *deletion to stars number* ( $\text{dts}$ ). Two vertices  $u, v \in V(G)$  have the *same type* if and only if  $N(v) \setminus \{u\} = N(u) \setminus \{v\}$ . A graph  $G$  has *neighborhood diversity* at most  $w$  if there exists a partition of  $V(G)$  into at most  $w$  sets, such that all the vertices in each set have the same type.

<sup>2</sup> The cop number of a disconnected graph is the sum of the cop numbers of its components; hence, we assume connectedness.

**CnR.** CnR is a two-player perfect information pursuit-evasion game played on a graph. One player is referred as *cop player* and controls a set of *cops*, and the other player is referred as *robber player* and controls a single *robber*. The game starts with the cop player placing each cop on some vertex of the graph, and multiple cops may simultaneously occupy the same vertex. Then, the robber player places the robber on a vertex. Afterwards, the cop player and the robber player make alternate moves, starting with the cop player. In the cop player move, the cop player, for each cop, either moves it to an adjacent vertex (along an edge) or keeps it on the same vertex. In the robber player move, the robber player does the same for the robber. For simplicity, we will say that the cops (resp., robber) move in a cop (resp., robber) move instead of saying that the cop (resp., robber) player moves the cops (resp., robber). Throughout, we denote the robber by  $\mathcal{R}$ .

A situation where one of the cops, say,  $\mathcal{C}$ , occupies the same vertex as  $\mathcal{R}$  is a *capture*. The cops win if they have a strategy to capture  $\mathcal{R}$ , and  $\mathcal{R}$  wins if it has a strategy to evade a capture indefinitely. A graph  $G$  is *k-copwin* if  $k$  cops have a winning strategy in  $G$ . The *cop number* of  $G$ , denoted  $c(G)$ , is the minimum  $k$  such that  $G$  is  $k$ -copwin. For brevity,  $G$  is said to be *copwin* if it is 1-copwin (i.e.  $c(G) = 1$ ). Given a graph  $G$  and  $k \in \mathbb{N}$ , CnR asks whether  $G$  is  $k$ -copwin.

If a cop  $\mathcal{C}$  occupies a vertex  $v$ , then  $\mathcal{C}$  *attacks*  $N[v]$ . A vertex  $u$  is *safe* if it is not being attacked by any cop. If  $\mathcal{R}$  is on a vertex that is not safe, then  $\mathcal{R}$  is *under attack*. We say that some cops *guard* a subgraph  $H$  of  $G$  if  $\mathcal{R}$  cannot enter  $H$  without getting captured by one of these cops in the next cop move. We shall use the following result:

► **Proposition 11** ([2]). *One cop can guard an isometric path  $P$  after a finite number of cop moves.*

Currently, the best known algorithm to decide whether  $G$  is  $k$ -copwin is by Petr et al. [50]:

► **Proposition 12** ([50]). *CnR is solvable in  $\mathcal{O}(kn^{k+2})$  time.*

**Variations of CnR.** Several variations of CnR have been studied in the literature, differing mainly in the rules of movements of agents, the definition of the capture, and the capabilities of the agents. We provide below the definitions of the games considered in this paper here. A detailed overview of these games is provided in the full version [29].

**Lazy CnR.** In LAZY CnR [47], the cops are *lazy*, i.e., at most one cop can move during the cops' turn. The cop number in this game is denoted by  $c_{\text{lazy}}(G)$ .

**Cops and Attacking Robber.** In COPS AND ATTACKING ROBBER [8], if on a robber's turn, there is a cop in its neighborhood, then  $\mathcal{R}$  can attack the cop and *eliminate* it from the game. However, if more than one cop occupy a vertex and  $\mathcal{R}$  attacks them, then only one of the cops gets eliminated, and  $\mathcal{R}$  gets captured by one of the remaining cops. Here, the cop number is denoted by  $c_{\text{attack}}(G)$ .

**Fully Active CnR.** In FULLY ACTIVE CnR [34], in a cop/robber move, each cop/robber must move to an adjacent vertex. Here, the cop number is denoted by  $c_{\text{active}}(G)$ .

**Surrounding CnR.** SURROUNDING CnR [15] differs in the definition of capture. Here, a cop and  $\mathcal{R}$  can occupy the same vertex during the game, but  $\mathcal{R}$  cannot end its turn by remaining at a vertex occupied by some cop. The cops win by *surrounding*  $\mathcal{R}$ , i.e., if  $\mathcal{R}$  occupies a vertex  $v$ , then there is a cop at each vertex  $u \in N(v)$ . The *surrounding cop number* for  $G$  is denoted as  $c_{\text{surround}}(G)$ .

**Cops and Fast Robber.** In COPS AND FAST ROBBER [27],  $\mathcal{R}$  can move faster than the cops. If  $\mathcal{R}$  has speed  $s$ , then it can move along a path with at most  $s$  edges not containing any cop. The minimum number of cops that ensures capture of a fast robber with speed  $s$  in a graph  $G$  is denoted by  $c_s(G)$ . For  $s \geq 2$ , deciding whether  $c_s(G) \leq k$  is W[2]-hard parameterized by  $k$  even when input graph  $G$  is restricted to be a split graph [27].

**CnR on Directed Graphs.** In the game of CNR on directed graphs [42, 13, 21], the players can only move along the orientation of the arcs.

**An XP Algorithm for Variants.** For graph searching, there is a standard technique to get an  $n^{\mathcal{O}(k)}$  time XP algorithm. This technique involves generating a *game graph* where each vertex represents a possible placement of all the agents (game states) on the vertices of  $G$ . Petr et al. [50] implemented this algorithm for CNR in  $\mathcal{O}(kn^{k+2})$  time. It is not difficult to see that this algorithm can be made to work for all the variants we discussed (by changing the rules to navigate between game states). We discuss these extensions in the full version [29].

► **Proposition 13.** *Any variant of CNR considered in this paper is solvable in  $\mathcal{O}(kn^{k+2})$  time.*

**Parameterized Complexity.** In the framework of parameterized complexity, each problem instance is associated with a non-negative integer, called a *parameter*. A parameterized problem  $\Pi$  is *fixed-parameter tractable* (FPT) if there is an algorithm that, given an instance  $(I, k)$  of  $\Pi$ , solves it in time  $f(k) \cdot |I|^{\mathcal{O}(1)}$  for some computable function  $f(\cdot)$ . Two instances  $I$  and  $I'$  are *equivalent* when  $I$  is a Yes-instance if and only if  $I'$  is a Yes-instance. A *compression* of a parameterized problem  $\Pi_1$  into a (possibly non-parameterized) problem  $\Pi_2$  is a polynomial-time algorithm that maps each instance  $(I, k)$  of  $\Pi_1$  to an equivalent instance  $I'$  of  $\Pi_2$  such that size of  $I'$  is bounded by  $g(k)$  for some computable function  $g(\cdot)$ . If  $g(\cdot)$  is polynomial, then the problem is said to admit a *polynomial compression*. A *kernelization algorithm* is a compression where  $\Pi_1 = \Pi_2$ . Here, the output instance is called a *kernel*. Let  $\Pi_1$  and  $\Pi_2$  be two parameterized problems. A *polynomial parameter transformation* from  $\Pi_1$  to  $\Pi_2$  is a polynomial-time algorithm that, given an instance  $(I, k)$  of  $\Pi_1$ , generates an equivalent instance  $(I', k')$  of  $\Pi_2$  such that  $k' \leq p(k)$ , for some polynomial  $p(\cdot)$ . It is well-known that if  $\Pi_1$  does not admit a polynomial compression, then  $\Pi_2$  does not admit a polynomial compression [17]. We refer to the books [17, 28] for details on parameterized complexity.

### 3 Bounding the Cop Number

We will use the following lemma to derive upper bounds on  $c(G)$  for several graph parameters.

► **Lemma 14** (\*). *Let  $G$  be a graph and let  $U \subseteq V(G)$  be a set of vertices such that for each connected component  $H$  of  $G[V(G) \setminus U]$ ,  $c(H) \leq \ell$ . Then,  $c(G) \leq \frac{|U|}{2} + \ell$ .*

Since star graphs and complete graphs are copwin, Lemma 14 implies the following theorem.

► **Theorem 15.** *Let  $G$  be a graph and  $t = \min\{\text{cvd}, \text{dts}\}$ . Then,  $c(G) \leq \frac{t}{2} + 1$ .*

**Bounding Cop Number by vcn.** Let  $U$  be a vertex cover of size  $t$  in  $G$  and  $I$  be the independent set  $V(G) \setminus U$ . Lemma 14 implies that  $c(G) \leq \lceil \frac{t}{2} \rceil + 1$ . In this section, we improve this bound. First, we provide the following reduction rules.

► **Reduction Rule 1 (RR1).** *If there is a vertex  $v \in I$  such that  $|N(v)| \geq 3$ , then place a cop at  $v$  and delete  $N[v]$ .*

► **Reduction Rule 2 (RR2).** *If there is a vertex  $v \in U$  such that  $|N[v] \cap U| \geq 3$ , then place a cop at  $v$  and delete  $N[v]$ .*

► **Reduction Rule 3 (RR3).** *If there is an isometric path  $P$  such that  $P$  contains at least three vertices from  $U$ , then guard  $P$  using one cop and delete  $V(P)$  (see Proposition 11).*

We note the following.

► **Note 16.** In the application of RR1-RR3, whenever a set of vertices  $X \subseteq V(G)$  is deleted by the application of RR1-RR3, it implies that each vertex  $x \in X$  is being guarded by some cop, and hence, is not accessible to  $\mathcal{R}$ . We do not actually delete the vertices, and this deletion part is just for the sake of analysis. Hence, from the cop player's perspective, the graph remains connected.

Next, we argue that, after an exhaustive application of RR1-RR3 on  $G$ , each connected component has a special structure and is 2-copwin.

► **Lemma 17 (\*).** *Once we cannot apply RR1-RR3 anymore, let  $\mathcal{R}$  be in a connected component  $H$  of  $G$ . Then,  $c(H) \leq 2$ .*

Finally, we have the following theorem.

► **Theorem 1.** *For a graph  $G$ ,  $c(G) \leq \frac{vcn}{3} + 1$ .*

**Proof Sketch.** The proof follows from Lemma 17 and the fact that in each application of RR1-RR3, we use one cop to remove (or guard) at least three new vertices from  $U$ . ◀

We note that a similar technique will fail if we try to “remove” four vertices in each reduction rule [29].

## 4 Exponential Kernels

**Exponential Kernel by  $vcn$ .** Let  $G$  be a graph where a vertex cover  $U$  of size  $t$  is given. If no such vertex cover is given, then we can compute a vertex cover  $U$  of size  $t \leq 2 \cdot vcn$  using a polynomial-time approximation algorithm [55]. Let  $I$  be the independent set  $V(G) \setminus U$ . Our kernelization algorithm is based on the exhaustive application of the following reduction rules.

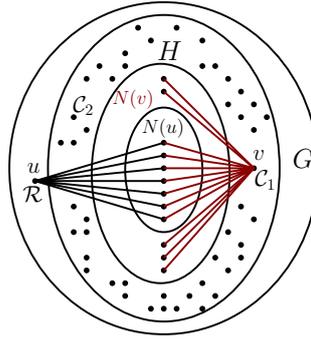
► **Reduction Rule 4 (RR4).** *If  $k > \frac{t}{3}$ , then answer positively.*

► **Reduction Rule 5 (RR5).** *If  $k = 1$ , then apply an  $\mathcal{O}(n^3)$  time algorithm (Proposition 12) to check whether  $G$  is copwin.*

► **Reduction Rule 6 (RR6).** *If there are two distinct vertices  $u, v \in I$  such that  $N(u) \subseteq N(v)$ , then delete  $u$ .*

The safeness of RR4 follows from Theorem 1. The following lemma proves the safeness of RR6. We note that Lemma 18 can also be derived from [6, Corollary 3.3], but we give a self-contained proof for the sake of completeness.

► **Lemma 18.** *Let  $u$  and  $v$  be two distinct vertices of  $G$  such that  $N(u) \subseteq N(v)$ . Consider the subgraph  $H$  of  $G$  induced by  $V(G) \setminus \{u\}$ . Let  $k \geq 2$ . Then,  $G$  is  $k$ -copwin if and only if  $H$  is  $k$ -copwin.*



■ **Figure 1** Illustration for Lemma 18. Here,  $\mathcal{R}$  is at vertex  $u$  and  $\mathcal{C}_1$  is at vertex  $v$ .

**Proof.** First, let  $G$  be  $k$ -copwin. Then, for the graph  $H$ , the  $k$  cops borrow the winning strategy that they have for  $G$ , with the only difference that whenever a cop has to move to the vertex  $u$  in  $G$ , it moves to  $v$  (in  $H$ ) instead. Since  $N(u) \subseteq N(v)$ , the cop can make the next move as it does in the winning cop strategy for  $G$ . Note that using this strategy, the cops can capture  $\mathcal{R}$  if  $\mathcal{R}$  is restricted to  $V(H)$  in  $G$ . Therefore, using this strategy,  $k$  cops will capture  $\mathcal{R}$  in  $H$  as well.

Second, we show that if  $H$  is  $k$ -copwin, then  $G$  is  $k$ -copwin. Here, for each vertex  $x \neq u$  of  $G$ , we define  $I(x) = x$ , and for  $u$ , we define  $I(u) = v$ . Observe that for each  $x \in V(G)$ ,  $I(x)$  is restricted to  $H$  and if  $xy \in E(G)$ , then  $I(x)I(y) \in E(H)$ . Therefore, every valid move of a player from a vertex  $x$  to  $y$  in  $G$  can be translated to a valid move from  $I(x)$  to  $I(y)$  in  $H$ . Now, the cops have the following strategy. If the robber is on a vertex  $x$ , the cops consider the *image* of the robber on the vertex  $I(x)$ . Since the robber’s image is restricted to  $H$ , the cops can use the winning strategy for  $H$  to capture the image of the robber in  $G$ . Once the image is captured, if the robber is not on the vertex  $u$ , then the robber is also captured. Otherwise, the robber is on the vertex  $u$ , and at least one cop is on  $v$ . See Figure 1 for an illustration. So, one cop, say  $\mathcal{C}_1$ , stays on  $v$  and this prevents the robber from ever leaving  $u$ . Indeed this follows because  $N(u) \subseteq N(v)$ , and so, if  $\mathcal{R}$  ever leaves  $u$ , it will be captured by  $\mathcal{C}_1$  in the next cop move. Finally, since  $k > 1$ , some other cop, say  $\mathcal{C}_2$ , can use a finite number of moves to reach  $u$  and capture the robber. ◀

Note that the requirement for  $k \geq 2$  in Lemma 18 is crucial. Otherwise, we can get an  $H$  such that  $c(H) = 1$ , but  $c(G) > 1$ . For example, consider  $C_4$ , where any two non-adjacent vertices satisfy the property in RR6, and if we remove one of them, the cop number reduces from 2 to 1. However, this does not harm our algorithm because if we are given  $k = 1$ , then RR5 is applied (before RR6).

Two sets  $A$  and  $B$  are *incomparable* if neither  $A \subseteq B$  nor  $B \subseteq A$ . We shall use the following proposition that follows from Sperner’s Theorem and Stirling’s approximation.

► **Proposition 19.** *Let  $X$  be a set of cardinality  $N$ . Moreover, let  $Y$  be a set of subsets of  $X$  such that for each  $a, b \in Y$ ,  $a$  and  $b$  are incomparable. Then,  $|Y| \leq \frac{2^N}{\sqrt{N}}$ .*

Once we cannot apply RR4-RR6 anymore, we claim that the size of the reduced graph  $G'$  is bounded by a function of  $t$ . Let  $U' = U \cap V(G')$  and  $I' = I \cap V(G')$ . Clearly,  $|U'| \leq t$ . Now, each vertex  $u \in I'$  is associated with a neighborhood  $N(u)$  such that  $N(u) \subseteq U'$ . Moreover, for any two vertices  $u, v \in I'$ , the sets  $N(u)$  and  $N(v)$  are incomparable. Hence, due to Proposition 19,  $|I'| \leq \frac{2^t}{\sqrt{t}}$ , and therefore,  $|V(G')| \leq t + \frac{2^t}{\sqrt{t}}$ , which proves the following theorem.

► **Theorem 2.** *CNR parameterized by  $\text{vcn}$  admits a kernel with at most  $\text{vcn} + \frac{2^{\text{vcn}}}{\sqrt{\text{vcn}}}$  vertices.*

The details for exponential kernels for other parameters and variants can be found in [29]. Here, we present only our reduction rules and the essential claims that lead to our results.

**Kernel by  $\text{cvd}$ .** Let  $U$  be a cluster vertex deletion set of size  $t$ . Let  $S = V(G) \setminus U$ , and  $C_1, \dots, C_\ell$  be the set of disjoint cliques in  $G[S]$ . We have the following reduction rules along with RR 5.

► **Reduction Rule 7 (RR7).** *If  $k > \frac{t}{2}$ , then answer positively.*

► **Reduction Rule 8 (RR8).** *Let  $u$  and  $v$  be vertices of some clique  $C \in G[S]$  such that  $N[u] \subseteq N[v]$ . Then, delete  $u$ .*

► **Reduction Rule 9 (RR9).** *Let  $C_i$  and  $C_j$  be two cliques in  $G[S]$  such that for each vertex  $u \in V(C_i)$ , there exists a vertex  $v \in V(C_j)$  such that  $N_U(u) \subseteq N_U(v)$ . Then, delete  $V(C_i)$ .*

RR7 is safe due to Theorem 15. The safeness of RR8 is proved by the following Lemma, whose proof is similar to the proof of Lemma 18.

► **Lemma 20 (\*).** *Let  $u$  and  $v$  be vertices of some clique  $C$  of  $G[S]$ . If  $N_U(u) \subseteq N_U(v)$ , then  $c(G) = c(G[V(G) \setminus \{u\}])$ .*

The safeness of RR9 is proved by the following Lemma.

► **Lemma 21 (\*).** *Let  $C_i$  and  $C_j$  be two cliques in  $G[S]$  such that for each vertex  $u \in V(C_i)$ , there exists a vertex  $v \in V(C_j)$  such that  $N_U(u) \subseteq N_U(v)$ . Then, for  $k > 1$ ,  $G$  is  $k$ -copwin if and only if  $G[V(G) \setminus V(C_i)]$  is  $k$ -copwin.*

Finally, we use the following lemma to bound the size of the desired kernel from Theorem 6.

► **Lemma 22 (\*).** *After an exhaustive application of RR7-RR9, the size of the reduced graph is at most  $2^{2^t + \sqrt{t}}$ .*

**Kernel by  $\text{dts}$ .** Using similar ideas, we can also get a kernel for CNR parameterized by  $\text{dts}$ . Let  $U$  be a deletion to stars vertex set of size  $t$ . Also, let  $S = V(G) \setminus U$ , and let  $X_1, \dots, X_\ell$  be the stars in the graph  $G[S]$ . Specifically, we have the following reduction rules along with RR7.

► **Reduction Rule 10 (RR10).** *Let  $u$  and  $v$  be two leaves of a star  $X$  in  $G[S]$  such that  $N_U(u) \subseteq N_U(v)$ . Then, delete  $u$ .*

► **Reduction Rule 11 (RR11).** *Let  $X$  and  $Y$  be two stars in  $G[S]$  such that  $V(X) = x, x_1, \dots, x_p$  and  $V(Y) = y, y_1, \dots, y_q$ , where  $x$  and  $y$  are center vertices of  $X$  and  $Y$ , respectively. If  $N_U(x) \subseteq N_U(y)$  and for each vertex  $x_i$  (for  $i \in [p]$ ), there is a vertex  $y_j$  (for  $j \in [q]$ ) such that  $N_U(x_i) \subseteq N_U(y_j)$ , then delete  $X$ .*

The following lemma establishes that RR10 and RR11 are safe. (RR7 is safe due to Theorem 15.)

► **Lemma 23 (\*).** *Assuming  $k > 1$ , RR10 and RR11 are safe.*

Using calculations similar to ones used in our previous kernels and Proposition 19, we can establish that (see [29]), once we cannot apply RR10 and RR11 anymore, the size of the reduced graph can be at most  $\frac{2^{2^t}}{\sqrt{2^t}} \cdot 2^t \cdot (\frac{2^t}{\sqrt{t}} + 1)$ , giving us the desired kernel from Theorem 6.

### Kernels for Different Variants

**Lazy CnR and Cops and Attacking Robber.** For these variants, we have the following reduction rules along with RR6, which we apply before applying RR6.

- ▶ **Reduction Rule 12** (RR12). *If  $k \geq \frac{t}{2} + 1$ , then answer positively (Theorem 7).*
- ▶ **Reduction Rule 13** (RR13). *If  $k = 1$ , then apply the  $\mathcal{O}(n^3)$  time algorithm from Proposition 13.*

The next lemma proves safeness of RR6 for both variants.

- ▶ **Lemma 24** (\*). *Let  $u$  and  $v$  be two distinct vertices of  $G$  such that  $N(u) \subseteq N(v)$ . Consider the graph  $H$  induced by  $V(G) \setminus \{u\}$ . Then for  $k > 1$  and for  $x \in \{\text{lazy, attack}\}$ ,  $c_x(G) \leq k$  if and only if  $c_x(H) \leq k$ .*

The size of the kernel, by using these reduction rules, is dependent on RR6. Therefore, we get the desired kernels for these variants as claimed in Theorem 8.

**CnR on Directed Graphs.** Next, we consider the game of CnR on directed graphs. We have the following lemma.

- ▶ **Lemma 25** (\*). *Let  $u$  and  $v$  be two distinct vertices of a strongly connected directed graph  $\vec{G}$  such that  $N^+(u) \subseteq N^+(v)$  and  $N^-(u) \subseteq N^-(v)$ . Let  $\vec{H}$  be the graph induced by  $V(\vec{G}) \setminus \{u\}$ . Then, for  $k > 1$ ,  $\vec{H}$  is  $k$ -copwin if and only if  $\vec{G}$  is  $k$ -copwin.*

Let  $G$  be a graph with a vertex cover  $U$  of size  $t$ , and let  $I = V(G) \setminus U$ . Let  $\vec{G}$  be a strongly connected orientation of  $G$ . We apply the following reduction rules.

- ▶ **Reduction Rule 14** (RR14). *If  $k \geq t$ , then answer positively.*
- ▶ **Reduction Rule 15** (RR15). *If  $k = 1$ , then apply the  $\mathcal{O}(n^3)$  time algorithm (Proposition 13) to check if  $\vec{G}$  is copwin.*
- ▶ **Reduction Rule 16** (RR16). *If  $u$  and  $v$  are two distinct vertices in  $I$  such that  $N^+(u) \subseteq N^+(v)$  and  $N^-(u) \subseteq N^-(v)$ , then delete  $u$ .*

Safeness of RR14 and RR16 follow from Theorem 7 and Lemma 25, respectively. Once we cannot apply RR16, observe that each vertex  $u \in I$  has a unique neighborhood ( $N^+(u) \cup N^-(u)$ ), and there are three choices for a vertex  $v \in U$  to appear in the neighborhood of  $u$ , i.e., either  $v \in N^+(u)$ , or  $v \in N^-(u)$ , or  $v \notin N^+(u) \cup N^-(u)$ . Thus,  $|I| \leq 3^t$ , giving the desired kernel as claimed in Theorem 8.

**General Kernelization.** Here, we provide a general reduction rule that works for most variants of CnR parameterized by vcn. Let  $U$  be a vertex cover of size  $t$  in  $G$  and  $I$  be the independent set  $V(G) \setminus U$ . For each subset  $S \subseteq U$ , we define the following equivalence class:  $\mathcal{C}_S = \{v \in I : N(v) = S\}$ . We have the following reduction rule along with RR14.

- ▶ **Reduction Rule 17** (RR17). *If there is an equivalence class  $\mathcal{C}_S$  such that  $|\mathcal{C}_S| > k + 1$ , then keep only  $k + 1$  arbitrary vertices from  $\mathcal{C}_S$  in  $G$ , and delete the rest.*

The following lemma proves the safeness of RR17.

► **Lemma 26** (\*). *Let  $G$  be a graph with a vertex cover  $U$  of size  $t$ . Let  $\mathcal{C}_S$  (for  $S \subseteq U$ ) be an equivalence class such that  $|\mathcal{C}_S| = \ell > k + 1$ . Moreover, let  $H$  be a subgraph formed by deleting  $\ell - k - 1$  arbitrary vertices of  $\mathcal{C}_S$  from  $G$ . Then,*

1.  $c_{\text{active}}(H) \leq k$  if and only if  $c_{\text{active}}(G) \leq k$ .
2.  $c_s(H) \leq k$  if and only if  $c_s(G) \leq k$ , for any  $s \geq 1$ .
3.  $c_{\text{surround}}(H) \leq k$  if and only if  $c_{\text{surround}}(G) \leq k$ .

**Proof Sketch.** Here, we present a sketch of the proof for (3). Let  $\mathcal{C}_S = \{v_1, \dots, v_\ell\}$ . WLOG, let the vertices  $v_1, \dots, v_{k+1}$  belong to the graph  $H$  and vertices  $v_{k+2}, \dots, v_\ell$  are deleted. Note that  $\mathcal{R}$  cannot be surrounded at a vertex in  $S$  in  $G$  since each vertex in  $S$  has at least  $k + 1$  neighbours.

The proof of  $c_{\text{surround}}(H) \leq c_{\text{surround}}(G)$  is similar to the proof of Lemma 18.

For the other direction, let  $c_{\text{surround}}(H) \leq k$ . Since we have only  $k$  cops, at any time, there is at least one vertex in  $\{v_1, \dots, v_{k+1}\}$  that is not occupied by any cop. Let us call this vertex a *free vertex* (there might be multiple free vertices). For each vertex  $x \in V(G)$ , if  $x \in V(H)$ , then we define  $I(x) = x$ ; else, if  $x \in \{v_{k+1}, \dots, v_\ell\}$ , then we define  $I(x) = y$ , where  $y$  is a free vertex at that instance. Whenever  $\mathcal{R}$  moves to a vertex  $x \in V(G)$ , we say that the *image of the robber*, denoted  $\mathcal{I}_{\mathcal{R}}$ , moves to  $I(x)$ . Recall that, in this variant, although some cop and  $\mathcal{R}$  can be at the same vertex,  $\mathcal{R}$  cannot end its move at the same vertex as one of the cops. Cops use this capability to force  $\mathcal{R}$  to move from a vertex. Therefore, we also have to argue that whenever cops force  $\mathcal{R}$  to move, they force  $\mathcal{I}_{\mathcal{R}}$  to move as well. To this end, observe that  $\mathcal{I}_{\mathcal{R}}$  and  $\mathcal{R}$  are on different vertices only if  $\mathcal{R}$  is on some vertex  $x \in \{v_{k+1}, \dots, v_\ell\}$  and  $\mathcal{I}_{\mathcal{R}}$  is on a free vertex, say,  $y$ . Notice that if, in the strategy for  $H$ ,  $\mathcal{R}$  was occupying  $y$  and the cop player wants to force  $\mathcal{R}$  to move out of  $y$ , then it does so by moving a cop, say,  $\mathcal{C}$ , from a vertex  $w \in N(y)$  to  $y$ . Cop player adapts this strategy in  $G$  by moving  $\mathcal{C}$  from  $w$  to  $x$  instead of  $w$  to  $y$  (it is possible because  $N(x) = N(y)$ ). Thus,  $\mathcal{R}$ , as well as  $\mathcal{I}_{\mathcal{R}}$ , are forced to move as they would have been forced to move in the winning strategy of  $k$  cops in  $H$ .

Hence,  $\mathcal{I}_{\mathcal{R}}$  is restricted to  $V(H)$  in  $G$ . Thus, cops will surround  $\mathcal{I}_{\mathcal{R}}$  in a finite number of rounds. The only thing to observe now is that if  $\mathcal{I}_{\mathcal{R}}$  is surrounded in  $V(H)$ , then  $\mathcal{R}$  is surrounded in  $G$ . ◀

Lemma 26 and the following lemma imply Theorem 9.

► **Lemma 27** (\*). *Let  $G$  be a graph with a vertex cover  $U$  of size  $t$ . After an exhaustive application of RR14 and RR17, the reduced graph has at most  $t + t \cdot 2^t$  vertices.*

## 5 Incompressibility

In this section, we show that it is unlikely for CNR parameterized by  $\text{vcn}$  to admit a polynomial compression. For this purpose, we first define the following problem. In RED-BLUE DOMINATING SET, we are given a bipartite graph  $G$  with a vertex bipartition  $V(G) = T \cup N$  and a non-negative integer  $k$ . A set of vertices  $N' \subseteq N$  is said to be an *RBDS* if each vertex in  $T$  has a neighbor in  $N'$ . The aim of RED-BLUE DOMINATING SET is to decide whether there exists an *RBDS* of size at most  $k$  in  $G$ . We shall use the following result.

► **Proposition 28** ([23]). *RED-BLUE DOMINATING SET parameterized by  $|T| + k$  does not admit a polynomial compression, unless  $NP \subseteq \text{coNP}/\text{poly}$ .*

We show that CNR parameterized by  $\text{vcn}$  does not have a polynomial compression by developing a polynomial parameter transformation from RED-BLUE DOMINATING SET parameterized by  $|T| + k$  to CNR parameterized by  $\text{vcn}$ .

**Bipartite Graphs with Large Degree and Girth.** For our reduction, we borrow a construction by Fomin et al. [27] of bipartite graphs having high girth and high minimum degree, which they used to prove NP-hardness (and  $W[2]$ -hardness for the solution size  $k$ ) of CNR. For positive integers  $p, q$ , and  $r$ , we can construct a bipartite graph  $H(p, q, r)$  with  $rqp^2$  edges and a bipartition  $(X, Y)$ , with  $|X| = |Y| = pq$ . The set  $X$  is partitioned into sets  $U_1, \dots, U_p$ , and the set  $Y$  is partitioned into sets  $W_1, \dots, W_p$ , with  $|U_i| = |W_i| = q$ . By  $H_{i,j}$  we denote the subgraph of  $H(p, q, r)$  induced by  $U_i \cup W_j$ , and by  $\deg_{i,j}(z)$  we denote the degree of vertex  $z$  in  $H_{i,j}$ . Fomin et al. [27] provided the following construction:

► **Proposition 29** ([27]). *Let  $q \geq 2p(r+1) \frac{(p(r+1)-1)^6-1}{(p(r+1)-1)^2-1}$ . Then, we can construct  $H(p, q, r)$  in time  $\mathcal{O}(r \cdot q \cdot p^2)$  with the following properties.*

1. *The girth of  $H(p, q, r)$  is at least 6.*
2. *For every vertex  $z \in V(H_{i,j})$  and every  $i, j \in [p]$ , we have  $r-1 \leq \deg_{i,j}(z) \leq r+1$ .*

**Polynomial Parameter Transformation.** Let  $(G, k)$  be an instance of RED-BLUE DOMINATING SET with  $V(G) = T \cup N$ . First, we construct a graph  $G'$  with  $V(G') = T' \cup N'$  from  $G$  by introducing two new vertices,  $x$  and  $y$ , such that  $T' = T \cup \{x\}$  and  $N' = N \cup \{y\}$ , and  $E(G') = E(G) \cup \{xy\}$ . We have the following observation.

► **Observation 30.**  *$G$  has an RBDS of size at most  $k$  if and only if  $G'$  has an RBDS of size at most  $k+1$ . Moreover, any RBDS of  $G'$  contains  $y$ .*

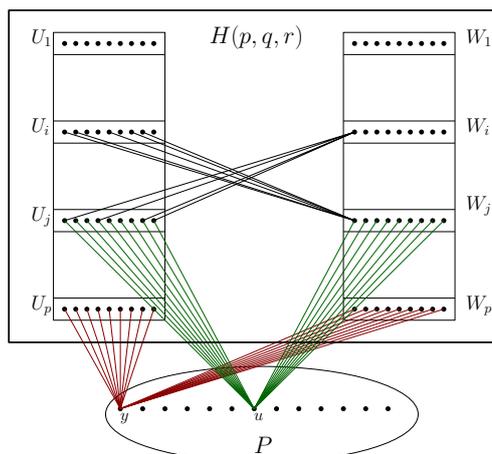
Now, we present the main construction for our reduction. Denote the vertex set  $V(T')$  by  $\{v_1, v_2, \dots, v_{p'}, x\}$ . Moreover, let  $p = p' + 1$ ,  $\ell = k + 1$ ,  $r = \ell + 2$ , and  $q = \lceil 2p(r+1) \frac{(p(r+1)-1)^6-1}{(p(r+1)-1)^2-1} \rceil$ .

We construct  $H(p, q, r)$  such that each of  $U_i$  and  $W_i$ , for  $0 < i \leq p'$ , contains  $q$  copies of vertex  $v_i$ , and each of  $U_p$  and  $W_p$  contains  $q$  copies of vertex  $x$ . Now, we obtain a graph  $G''$  by adding one more set of vertices  $P$  to  $H(p, q, r)$  such that  $V(P) = V(N')$ . Moreover, if there is an edge between a vertex  $u \in N'$  and a vertex  $v_i \in T'$ , then we add an edge between  $u$  and every vertex of  $U_i$ , and also between  $u$  and every vertex of  $W_i$ . Similarly, we add an edge between  $y$  and every vertex of  $U_p$ , and between  $y$  and every vertex of  $W_p$ . Finally, we make the vertex  $y$  adjacent to every vertex of  $P$ . See Figure 2 for reference. For correctness, we have the following lemma.

► **Lemma 31.**  *$G'$  has an RBDS of size at most  $\ell$  if and only if  $G''$  is  $\ell$ -copwin.*

**Proof.** First, we show that if  $G'$  has an RBDS of size  $\ell$ , then  $\ell$  cops have a winning strategy in  $G''$ . Let  $S \subseteq N'$  be an RBDS in  $G'$  of size at most  $\ell$ . The cops begin by choosing the vertices corresponding to  $S$  in  $P$ . Observe that the vertex  $y$  has to be present in  $S$ . Since vertex  $y$  dominates each vertex in  $P$ , the robber cannot safely enter a vertex in  $P$ . Additionally, due to the construction of  $G''$ , the vertices of  $S$  dominate each vertex in  $H$ . Hence, the robber cannot safely enter a vertex in  $H$ . Therefore, the robber will be captured in the first move of the cops.

Next, we show that if there is no RBDS of size  $\ell$  in  $G'$ , then  $\ell$  cops do not have a winning strategy. We prove this by giving a winning strategy for the robber. First, we show that the robber can safely enter the graph. In the beginning, let there be  $\ell_1 \leq \ell$  cops in  $P$  and  $\ell_2 \leq \ell$  cops in  $H$ . Since there is no RBDS of size  $\ell$  in  $G'$ , for every placement of at most  $\ell$  cops in  $P$ , there exists at least one pair of  $U_i$  and  $W_i$  such that no vertex of  $U_i$  and  $W_i$  is dominated by the cops from  $P$ . Let  $U_i$  and  $W_i$  be one such pair of sets such that no vertex of  $U_i$  and  $W_i$  is dominated by the cops from  $P$ . Moreover, since each vertex of  $H$  can dominate at



■ **Figure 2** Illustration for  $H(p, q, r)$  and  $P$ . If a vertex  $u$  is connected to  $v_j$  in  $G$ , then  $u$  is connected to every vertex of  $W_j \cup U_j$ . Moreover, for every  $i, j$ , each vertex in  $U_i$  has at least  $r - 1$  neighbors in  $U_j$ .

most  $p(r + 1)$  vertices in  $H$ ,  $\ell_2$  cops can dominate at most  $\ell \cdot p(r + 1)$  vertices. Since  $U_i$  (and  $W_i$  also) contains  $q$  vertices, and  $q > \ell \cdot p(r + 1)$ , the  $\ell_2$  cops in  $H$  cannot dominate all vertices of  $U_i$ , and hence the robber can safely enter a vertex of  $U_i$ .

Now, whenever the robber is under attack, it does the following. Without loss of generality, let us assume that the robber is in  $U_i$  (the case of  $W_i$  is symmetric). Since there are at most  $\ell$  cops in  $P$ , there is always a  $W_j$  such that no vertex of  $W_j$  is dominated by cops from  $P$ . Since each vertex in  $U_i$  has at least  $r - 1 = \ell + 1$  neighbours in  $W_j$ , the robber can move to at least  $\ell + 1$  vertices of  $W_j$ . Since the girth of  $H$  is at least 6, no vertex from  $H$  can dominate two vertices of  $W_j$  that are adjacent to the robber; else, we get a cycle on four vertices. Hence, at most  $\ell$  cops from  $H$  can dominate at most  $\ell$  neighbors of the robber in  $W_j$ , and the robber has at least  $\ell + 1$  neighbors in  $W_j$ . Hence, the robber can move to a safe vertex in  $W_j$ . Since the graph  $H$  is symmetric, the robber can move safely from  $W_j$  to  $W_{j'}$  also. The robber follows this strategy to avoid capture forever. ◀

Next, we have the following observation to show that there exists a vertex cover  $U$  of  $G''$  such that  $|U| = \text{poly}(|T|, k)$ .

► **Observation 32.**  $V(H) \cup \{y\}$  is a vertex cover of  $G''$ . Therefore, the vertex cover number of  $G''$  is at most  $2 \cdot p \cdot q + 1 = 1 + 2p \cdot \lceil 2p(k + 3) \frac{(p(k+3)-1)^6 - 1}{(p(k+3)-1)^2 - 1} \rceil$ , where  $p = |T| + 1$ .

This completes the proof of the argument that CNR parameterized by vcn is unlikely to admit a polynomial compression. Thus, we have the following theorem as a result of Lemma 31, Observation 32 and Proposition 28.

► **Theorem 4.** CNR parameterized by vcn does not admit polynomial compression, unless  $NP \subseteq \text{coNP}/\text{poly}$ .

## 6 Conclusion and Future Directions

To achieve our kernelization results, the rules we used concerned removing (false or true) twins from the graph. These rules are easy to implement and hence can be used to reduce the complexity of the input graph, even when the input graph is far from the considered

parameters. For example, for cographs and grids, none of the considered parameters is constant/bounded, but cographs and grids can be reduced to a single vertex with the operation of removing twins, and hence, our reduction rules give an alternate proof that the cop number of cographs and grids is at most two [38, 19] for several variants. Moreover, MTP is well-studied with the motivation of designing computer games. Some examples of these variants include: multiple targets and multiple pursuer search [56] with applications in controlling non-player characters in video games; MTP from the robber’s perspective with faster cops [44] where the strategies were tested on Baldur’s Gate; MTP modeled with edge weights and different speeds of agents [36] with the specific examples of Company of Heroes and Supreme Commander. Moreover, the PACMAN game’s movement can be considered as an instance of FULLY ACTIVE CNR on a partial grid. One of the key aspects of designing these games is to come up with scenarios that are solvable but look complex and challenging. Our reduction rule can there. One can begin with an easy-to-resolve instance of CNR, and then keep adding twins to this instance (recursively) to get an instance that looks sufficiently complex but has the same complexity.

Finally, we define (formally defined in the full version [29]) a new variant of CNR, named GENERALIZED CNR, that generalizes many well-studied variants of CNR. Here the input is  $(G, \mathcal{C}_1, \dots, \mathcal{C}_k, \mathcal{R})$  where each cop  $\mathcal{C}_i$  has *speed*  $s_i$  and  $\mathcal{R}$  has speed  $s_R$ . Moreover, each cop can be either forced to be *active* (have to move in each turn), *lazy* (at most one lazy cop moves in each turn), or *flexible*. Furthermore, each cop  $\mathcal{C}_i$  can have *reach*  $\lambda_i$ . (Think of  $\mathcal{C}_i$  having a gun with range  $\lambda_i$ , and if  $\mathcal{R}$  is ever at a vertex that is at a distance at most  $\lambda_i$  from  $\mathcal{C}_i$ , it gets shot.) In the full version [29], we show that RR17 provides a kernel for GENERALIZED CNR as well. This gives hope that RR17 can be used to get kernels for many practical variants not explicitly studied in this paper. Also, RR17 has been used to provide kernelization algorithm for HUNTER AND RABBIT game parameterized by  $\text{vcn}$  [22].

Still, many questions on the parameterized complexity of CNR remain open. We list some of these questions below.

► **Question 33.** *Does there exist an FPT algorithm for CNR parameterized by  $\text{vcn}$  with running time  $2^{\mathcal{O}(\text{vcn})} \cdot n^{\mathcal{O}(1)}$ ?*

► **Question 34.** *Does there exist a better bound for the cop number with respect to  $\text{vcn}$ ? In particular, is  $c(G) = o(\text{vcn})$ ?*

► **Question 35.** *Does CNR parameterized by  $\text{vcn}$  admit a polynomial  $\alpha$ -approximate kernel?*

► **Question 36.** *Study CNR with respect to the following parameters: (1) feedback vertex set (2) treewidth (3) treedepth. In particular, is CNR FPT parameterized by treewidth?*

---

## References

- 1 I. Abraham, C. Gavaille, A. Gupta, O. Neiman, and K. Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing*, 48(3):1120–1145, 2019.
- 2 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.
- 3 T. Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory Series B*, 41(1):37–47, 1986.
- 4 D. Angelo, A. Navarra, and N. Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing*, 30:17–48, 2017.

- 5 J. A. Baier, A. Botea, D. Harabor, and C. Hernández. Fast algorithm for catching a prey quickly in known and partially known game maps. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):193–199, 2015. doi:10.1109/TCIAIG.2014.2337889.
- 6 A. Berarducci and B. Intrigila. On the cop number of a graph. *Advances in Applied mathematics*, 14(4):389–403, 1993.
- 7 A. Beveridge, A. Dudek, A. Frieze, and T. Müller. Cops and robbers on geometric graphs. *Combinatorics, Probability and Computing*, 21(6):816–834, 2012.
- 8 A. Bonato, S. Finbow, P. Gordinowicz, A. Haidar, W. B. Kinnersley, D. Mitsche, P. Prałat, and L. Stacho. The robber strikes back. In *proceedings of Computational Intelligence, Cyber Security and Computational Models*, volume 246, pages 3–12. Springer, 2014.
- 9 A. Bonato and R. Nowakowski. *The Game of Cops and Robbers on Graphs*. American Mathematical Society, 2011.
- 10 E. Bonnet, S. Gaspers, A. Lambilliotte, S. Rümmele, and A. Saffidine. The parameterized complexity of positional games. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, pages 90:1–90:14, 2017.
- 11 E. Bonnet, F. Jamain, and A. Saffidine. On the complexity of connection games. *Theoretical Computer Science*, 644:2–28, 2016.
- 12 N. Bowler, J. Erde, F. Lehner, and M. Pitz. Bounding the cop number of a graph by its genus. *SIAM Journal on Discrete Mathematics*, 35(4):2459–2489, 2021.
- 13 P. Bradshaw, S. A. Hosseini, and J. Turcotte. Cops and robbers on directed and undirected abelian Cayley graphs. *European Journal of Combinatorics*, 97, 2021.
- 14 S. Brandt, S. Pettie, and S. Uitto. Fine-grained lower bounds on cops and robbers. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.9.
- 15 A. C. Burgess, R. A. Cameron, N. E. Clarke, P. Danziger, S. Finbow, C. W. Jones, and D. A. Pike. Cops that surround a robber. *Discrete Applied Mathematics*, 285:552–566, 2020.
- 16 T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous Robots*, 31(299):299–316, 2011.
- 17 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.
- 18 J. Czyzowicz, L. Gasieniec, T. Gorry, E. Kranakis, R. Martin, and D. Pajak. Evacuating robots via unknown exit in a disk. In F. Kuhn, editor, *Distributed Computing (DISC 2014)*, volume 8784 of *Lecture Notes in Computer Science*, pages 122–136, 2018. doi:10.1007/978-3-662-45174-8\_9.
- 19 S. Das and H. Gahlawat. Variations of cops and robbers game on grids. *Discrete Applied Mathematics*, 305:340–349, 2021.
- 20 S. Das and H. Gahlawat. On the Cop Number of String Graphs. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2022.45.
- 21 S. Das, H. Gahlawat, U. k. Sahoo, and S. Sen. Cops and robber on some families of oriented graphs. *Theoretical Computer Science*, 888:31–40, 2021.
- 22 T. Dissaux, Fioravantes F., H. Gahlawat, and N. Nicolas. Recontamination helps a lot to hunt a rabbit. In Bérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 40:1–40:14, Dagstuhl, Germany, 2023.

- 23 M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and ids. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP '09*, pages 378–389, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-02927-1\_32.
- 24 H. E. Dudeney. *Amusements in Mathematics*. Facsimile Publisher, 1917.
- 25 J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.
- 26 M. R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings, IWPEC'06*, pages 276–277, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11847250\_25.
- 27 F. Fomin, P. Golovach, J. Kratochvíl, N. Nisse, and K. Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 41(7-9):1167–1181, 2010.
- 28 F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.
- 29 H. Gahlawat and M. Zehavi. Parameterized analysis of the cops and robber problem. *arXiv:2307.04594*, 2023. doi:10.48550/arXiv.2307.04594.
- 30 N. Galesi, N. Talebanfard, and J. Torán. Cops-robber games and the resolution of tseitin formulas. *ACM Transactions on Computation Theory*, 12(2):1–22, 2020.
- 31 G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.
- 32 G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.
- 33 G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, 2003.
- 34 I. Gromovikov, W. B. Kinnersley, and B. Seamone. Fully active cops and robbers. *Australian Journal of Combinatorics*, 76(2):248–265, 2020.
- 35 R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley and Sons, New York, 1965.
- 36 A. Isaza, J. Lu, V. Bulitko, and R. Greiner. A cover-based approach to multi-agent moving target pursuit. In *proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 54–59. AAAI Press, 2008.
- 37 T. Ishida and R. E. Korf. Moving target search. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI'91*, pages 204–210, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.
- 38 G. Joret, M. Kaminski, and D. O. Theis. Cops and robber game on graphs with forbidden (induced) subgraphs. *Contributions to Discrete Mathematics*, 5(2):40–51, 2010.
- 39 W. B. Kinnersley. Cops and robbers is exptime-complete. *Journal of Combinatorial Theory Series B*, 111:201–220, 2015.
- 40 K. Klein and S. Suri. Catch me if you can: Pursuit and capture in polygonal environments with obstacles. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2012)*, volume 26, pages 2010–2016, 2012. doi:DOI:10.1609/aaai.v26i1.8375.
- 41 F. Lehner. On the cop number of toroidal graphs. *Journal of Combinatorial Theory, Series B*, 151:250–262, 2021.
- 42 P. Loh and S. Oh. Cops and robbers on planar directed graphs. *Journal of Graph Theory*, 86(3):329–340, 2017.
- 43 M. Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477:48–56, 2013.
- 44 C. Moldenhauer and N. R. Sturtevant. Evaluating strategies for running from the cops. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, pages 584–589, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

- 45 N. Nisse. Network decontamination. In *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 516–548. Springer, 2019.
- 46 R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.
- 47 D. Offner and K. Ojakian. Variations of cops and robber on the hypercube. *Australasian Journal of Combinatorics*, 59(2):229–250, 2014.
- 48 T. D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs, LNCS*, volume 642, pages 426–441. Springer, 1978.
- 49 V. Patsko, S. Kumkov, and V. Turova. Pursuit-evasion games. In *Handbook of Dynamic Game Theory*, pages 951–1038. Springer, 2018.
- 50 J. Petr, J. Portier, and L. Versteegen. A faster algorithm for cops and robbers. *arXiv*, 2021. [arXiv:2112.07449v2](https://arxiv.org/abs/2112.07449v2).
- 51 A. Quilliot. *Thèse d'Etat*. PhD thesis, Université de Paris VI, 1983.
- 52 A. Scott. *On the Parameterized Complexity of Finding Short Winning Strategies in Combinatorial Games*. PhD thesis, University of Victoria, 2009.
- 53 A. Scott and U. Stege. Parameterized pursuit-evasion games. *Theoretical Computer Science*, 411(43):3845–3858, 2010.
- 54 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.
- 55 D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: [http://www.cambridge.org/de/knowledge/isbn/item5759340/?site\\_locale=de\\_DE](http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE).
- 56 F. Xie, A. Botea, and A. Kishimoto. A scalable approach to chasing multiple moving targets with multiple agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI'17*, pages 4470–4476. AAAI Press, 2017.



# An FPT Algorithm for Spanning Trees with Few Branch Vertices Parameterized by Modular-Width

Luisa Gargano

Department of Computer Science, University of Salerno, Italy

Adele A. Rescigno

Department of Computer Science, University of Salerno, Italy

---

## Abstract

---

The minimum branch vertices spanning tree problem consists in finding a spanning tree  $T$  of an input graph  $G$  having the minimum number of branch vertices, that is, vertices of degree at least three in  $T$ . This  $NP$ -hard problem has been widely studied in the literature and has many important applications in network design and optimization. Algorithmic and combinatorial aspects of the problem have been extensively studied and its fixed parameter tractability has been recently considered. In this paper we focus on modular-width and show that the problem of finding a spanning tree with the minimum number of branch vertices is FPT with respect to this parameter.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms; Mathematics of computing  $\rightarrow$  Graph theory

**Keywords and phrases** Spanning Trees, Branch vertices, Fixed-parameter tractable algorithms, Modular-width

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.50

**Funding** This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

## 1 Introduction

Let  $G = (V, E)$  be an undirected graph where  $V$  is the set of vertices and  $E$  is the set of edges. Given a spanning tree  $T$  of  $G$ , a *branch vertex* is a vertex having degree at least three in  $T$ . If  $G$  is a connected graph, we denote by  $b(G)$  the smallest number of branch vertices in any spanning tree of  $G$ . We study the following problem:

MINIMUM BRANCH VERTICES (MBV)

**Instance:** A connected graph  $G = (V, E)$ .

**Goal:** Find a spanning tree of  $G$  having  $b(G)$  branch vertices.

Notice that a spanning tree of  $G$  without branch vertices is a Hamilton path, that is,  $b(G) = 0$  if and only if  $G$  admits a Hamilton path.

The problem of determining a spanning tree with a bounded number of branch vertices, while a natural theoretical question, was introduced to solve a problem related to wavelength-division multiplexing technology in optical networks, where one wants to minimize the number of light-splitting switches in a light-tree [11]. Also for Cognitive Radio Networks other than for 5G technologies, that operate with a wide range of frequencies, bounding the switching costs due to the switching between different service providers has high importance both in terms of delay and energy consumption [16, 24]. MBV has been then widely studied, both from the algorithmic and the graph-theoretic point of view. Gargano *et al.* [12] proved that it is NP-complete to decide whether a graph  $G$  admits a spanning tree with at most  $k$  branch vertices, for given  $G$  and  $k$ , even in cubic graphs. Salamon [23] proved the existence of an algorithm that finds a spanning tree with  $O(\log |V(G)|)$  branch vertices whenever the



© Luisa Gargano and Adele A. Rescigno;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 50; pp. 50:1–50:15

Leibniz International Proceedings in Informatics



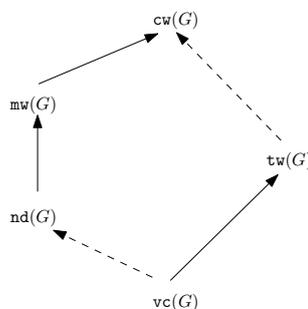
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

degree of each vertex of the input graph is  $\Omega(n)$ ; moreover, an approximation factor better than  $O(\log |V(G)|)$  would imply that  $NP \subseteq DTIME(n^{O(\log \log n)})$ . Sufficient conditions for a connected claw-free graph to have a spanning tree with  $k$  branch vertices are given in [20]. Integer linear formulations of MBV and some variants are presented in [5–7], together with different relaxations of them; the authors also provide numerical result comparison of the considered relaxations. In [26] hybrid integer linear programs for MB are considered and solved with branch-and-cut algorithms. In [18, 21, 22] decomposition methods of graphs are used to solve the MBV problem. Other heuristics are presented in [19, 25, 27]. In [8] a complementary formulation of MBV is investigated. It is called maximum path-node spanning tree (MPN), where the goal is to find a spanning tree that maximizes the number of vertices with degree at most two; the authors prove that MPN is APX-hard and present an approximation algorithm with ratio 6/11. Related gathering processes are considered in [2–4, 13, 14].

## 1.1 Parameterized Complexity

Parameterized complexity is a refinement to classical complexity theory in which one takes into account not only the input size, but also other aspects of the problem given by a parameter  $p$ . A problem with input size  $n$  and parameter  $p$  is called *fixed parameter tractable* (FPT) if it can be solved in time  $f(p) \cdot n^c$ , where  $f$  is a computable function only depending on  $p$  and  $c$  is a constant.

In this paper we are interested in assessing the complexity of MBV when parameterized by modular-width. It was recently proven that MBV is FPT when parameterized either by treewidth [1] or by neighborhood diversity [15]. On the other hand, it was shown in [9] that the problem is  $W[1]$ -hard when parameterized by clique-width. Specifically, in [9] it was proven that the (MBV special case) hamiltonian path problem is  $W[1]$ -hard when parameterized by clique-width. See Figure 1 for a relation among the above parameters.



■ **Figure 1** A summary of the relations holding among some popular parameters. We use  $mw(G)$ ,  $tw(G)$ ,  $cw(G)$ ,  $nd(G)$ , and  $vc(G)$  to denote modular-width, treewidth, cliquewidth, neighborhood diversity, and minimum vertex cover of a graph  $G$ , respectively. Solid arrows denote generalization, e.g., modular-width generalizes neighborhood diversity. Dashed arrows denote that the generalization may exponentially increase the parameter.

## 1.2 Modular-width

Modular-width was introduced in [10] as graph parameter which could cover dense graphs but still allows FPT algorithms for the problems lost to clique-width.

► **Definition 1** (Modular-width [10]). Consider graphs that can be obtained from an algebraic expression that uses the following operations:

- (O1) Create an isolated vertex;
- (O2) the disjoint union of 2 graphs denoted by  $G_1 \oplus G_2$ , i.e.,  $G_1 \oplus G_2$  is the graph with vertex set  $V(G_1) \cup V(G_2)$  and edge set  $E(G_1) \cup E(G_2)$ ;
- (O3) the complete join of 2 graphs denoted by  $G_1 \otimes G_2$ , i.e.,  $G_1 \otimes G_2$  is the graph with vertex set  $V(G_1) \cup V(G_2)$  and edge set  $E(G_1) \cup E(G_2) \cup \{\{v, w\} : v \in V(G_1) \text{ and } w \in V(G_2)\}$ ;
- (O4) the substitution operation with respect to some graph  $G$  with vertex set  $\{1, 2, \dots, n\}$  i.e., for graphs  $G_1, \dots, G_n$  the substitution of the vertices of  $G$  by the graphs  $G_1, \dots, G_n$ , denoted by  $G(G_1, \dots, G_n)$ , is the graph with vertex set  $\bigcup_{i=1}^n V(G_i)$  and edge set  $\bigcup_{i=1}^n E(G_i) \cup \{\{u, v\} \mid u \in V(G_i), v \in V(G_j), \{i, j\} \in E(G)\}$ . Hence,  $G(G_1, \dots, G_n)$  is obtained from  $G$  by substituting every vertex  $i \in V(G)$  with the graph  $G_i$  and adding all edges between the vertices of a graph  $G_i$  and the vertices of a graph  $G_j$  whenever  $\{i, j\} \in E(G)$ .

Let  $A$  be an algebraic expression that uses only the operations (O1)–(O4). The width of  $A$  is the maximum number of operands used by any occurrence of the operation (O4) in  $A$ . The modular-width of a graph  $H$ , denoted  $\text{mw}(H)$ , is the least integer  $m$  such that  $H$  can be obtained from such an algebraic expression of width at most  $m$ .

We recall that an algebraic expression of width  $\text{mw}(G)$  can be constructed in linear time [28].

Given a graph  $H = G(G_1, \dots, G_n)$ , we will refer to the graphs  $G_1, \dots, G_n$  also as the modules of  $H$ . Notice that given the graph  $H = G(G_1, \dots, G_n)$ , by the operations O(1)–(O4), one has that all the vertices of  $G_i$  share the same neighborhood outside  $G_i$ ; indeed,

$$\begin{aligned} \{\{u, v\} \mid u \in V(G_i), v \in V(G_j)\} \subseteq E(H) & \quad \text{if } \{i, j\} \in E(G) \\ \{\{u, v\} \mid u \in V(G_i), v \in V(G_j)\} \cap E(H) = \emptyset & \quad \text{if } \{i, j\} \notin E(G) \end{aligned} \tag{1}$$

for each  $i, j = 1, \dots, n$  with  $i \neq j$ .

### 1.3 Graph Partitioning

A *spider* is a tree with at most one branch vertex. The *center* of the spider is the branch vertex, if it exists, and is any vertex otherwise. A *path-spider cover* of a graph  $G$  is a set composed by one spider and some paths that are pair-wise (vertex-)disjoint and whose union contains every vertex of  $G$ . We denote by  $\text{spi}(G)$  the least integer  $p$  such that  $G$  has a path-spider cover with  $p - 1$  paths.

In order to solve MBV, we define and study the following problem that can be of its own interest:

PATH-SPIDER COVER (PSC)

**Instance:** A graph  $G = (V, E)$ .

**Goal:** Find a path-spider cover of  $G$  with  $\text{spi}(G) - 1$  paths.

Moreover, we will need the following Partitioning into Paths problem that was proven to be FPT with respect to modular-width in [10]. A *partition of a graph  $G$  into paths* is a set of (vertex-)disjoint paths of  $G$  whose union contains every vertex of  $G$ . We denote by  $\text{ham}(G)$  the least integer  $p$  such that  $G$  has a partition into  $p$  paths. Notice that  $\text{spi}(G) \leq \text{ham}(G)$ .

PARTITIONING INTO PATHS (PP)

**Instance:** A graph  $G = (V, E)$ .

**Goal:** Find a partition of  $G$  into  $\text{ham}(G)$  paths.

As originally defined in [10], the PARTITIONING INTO PATHS problem only asks for the value  $\text{ham}(G)$ , while we ask for the actual path partitioning of  $G$ .

In the following we will denote by  $\mathcal{P}_{ham(G)}$  a partition of  $G$  into  $ham(G)$  paths, and by  $\mathcal{P}_{spi(G)}$  a path-spider cover of  $G$  with  $spi(G) - 1$  paths.

Given a path  $P$  in  $G$ , we will denote by  $f(P)$  and  $s(P)$  the two end-points of  $P$ ; we will distinguish them as *the first and the second end-point of  $P$* , respectively. Furthermore, if  $P$  denotes a spider in  $G$  then we will equally use either  $f(P)$  or  $s(P)$  to denote *the center of  $P$* .

## 2 Our Results

We present an FPT algorithm for MBV parameterized by modular-width. To this aim, we also design a FPT algorithm for PSC parameterized by modular-width.

Let  $H$  be the input graph. Consider the parse-tree of an algebraic expression describing  $H$ , according to the rules (O1)-(O4) in Section 1.3. We take a look at the operation corresponding to the root: Operation (O1) is trivial and (O2) yields a disconnected graph, therefore we suppose the last operation is either (O3) or (O4). Hence, we can see the input graph as  $H = G(G_1, \dots, G_n)$  where  $G$  is a graph with  $n \leq mw(H)$  vertices and  $G_1, \dots, G_n$  are graphs.

The algorithm that finds a spanning tree of an input graph  $H$  with  $b(H)$  branch vertices goes through the following steps 1) and 2).

- 1) An FPT algorithm for PSC and PP parameterized by the modular-width of the input graph  $H$ . Namely, for each vertex  $\hat{H} = \hat{G}(\hat{G}_1, \dots, \hat{G}_n)$  of the parse-tree of  $H$ , we show how to compute the triple

$$(ham(\hat{H}), spi(\hat{H}), |V(\hat{H})|) \text{ together with } \mathcal{P}_{ham(\hat{H})} \text{ and } \mathcal{P}_{spi(\hat{H})},$$

where  $\mathcal{P}_{ham(\hat{H})}$  is a partition of  $\hat{H}$  into  $ham(\hat{H})$  paths and  $\mathcal{P}_{spi(\hat{H})}$  is a path-spider cover of  $\hat{H}$  with  $spi(\hat{H}) - 1$  paths.

- 2) Compute a spanning tree of  $H = G(G_1, \dots, G_n)$  with  $b(H)$  branch vertices by using the values, computed at step 1), for the graphs  $G_1, \dots, G_n$ , that is,

$$(ham(G_i), spi(G_i), |V(G_i)|), \mathcal{P}_{ham(G_i)}, \text{ and } \mathcal{P}_{spi(G_i)},$$

for  $i = 1, \dots, n$ ,

The computation in step 2) is only done once, i.e., for the root vertex of the parse tree, corresponding to the input graph  $H = G(G_1, \dots, G_n)$ . It is shown in Section 3, which is devoted to prove the following theorem.

► **Theorem 2.** MINIMUM BRANCH VERTICES *parameterized by modular-width is fixed-parameter tractable.*

The computation in step 1) is presented in Section 4. Following [10], we use a bottom-up dynamic programming approach along the parse-tree to compute for every vertex a record of data, using those already computed for its children. Since the operations of type (O1)-(O3) can be replaced by one operation of type (O4) that uses at most 2 operands, we only focus on the computation (and the time it requires) of a record of data for a vertex of type (O4) in the parse-tree. Namely, in Section 4 we prove the following theorem.

► **Theorem 3.** PATH-SPIDER COVER *parameterized by modular-width is fixed-parameter tractable.*

### 3 The MBV algorithm

In this section we give an algorithm that finds a spanning tree of graph  $H$  with  $b(H)$  branch vertices. We assume here that for each vertex  $\hat{H} = \hat{G}(\hat{G}_1, \dots, \hat{G}_n)$  of the parse-tree of  $H$ , we already have

the triple  $(\text{ham}(\hat{H}), \text{spi}(\hat{H}), |V(\hat{H})|)$  together with  $\mathcal{P}_{\text{ham}(\hat{H})}$  and  $\mathcal{P}_{\text{spi}(\hat{H})}$ .

We start by giving a characterization of a spanning tree with the minimum number of branch vertices in terms of the modular decomposition of  $H$ .

► **Lemma 4.** *Let  $H = G(G_1, \dots, G_n)$  be a connected graph. There exists a spanning tree of  $H$  with  $b(H)$  branch vertices that has at most one branch vertex belonging to  $G_i$  for each  $i = 1, \dots, n$ . Hence,  $b(H) \leq n \leq \text{mw}(H)$ .*

**Proof.** Let  $T$  be a spanning tree of  $H$  with  $b(H)$  branch vertices. Denote by  $B$  the set of vertices of  $H$  that are branch vertices in  $T$  and by  $N_T(v)$  the set of neighbors of  $v$  in  $T$ , for any vertex  $v$ . Assume that  $|V(G_i) \cap B| \geq 2$  for some  $i \in \{1, \dots, n\}$ . We show how to transform  $T$  so to satisfy the lemma. The transformation consists of two phases.

■ **Phase 1.** For each  $i = 1, \dots, n$ , we denote by  $B_i$  the set of branch vertices in  $V(G_i)$  that have in  $T$  at least two neighbors outside  $V(G_i)$ , that is,

$$B_i = \{v \mid v \in V(G_i) \cap B \text{ and } |N_T(v) \cap (\cup_{j \neq i} V(G_j))| \geq 2\}.$$

In this phase we transform  $T$  so that  $|B_i| \leq 1$ , for each  $i$ . We proceed as follows.

For each  $i$  such that  $|B_i| \geq 2$ ,

- let  $v$  be any node in  $B_i$ ;
- for each  $w \in B_i$  with  $w \neq v$ , consider the path connecting  $v$  and  $w$  in  $T$ , say  $v, \dots, w', w$ , and modify  $T$  as follows: For any  $x \in (N_T(w) - V(G_i)) - \{w'\}$ , substitute in  $T$  the edge  $\{w, x\}$  by the edge  $\{v, x\}$ .  
(Notice that this is possible by (1) and implies  $B_i = \{v\}$ ).

■ **Phase 2.** We know that each  $G_i$  contains at most one branch vertex with at least two neighbors outside  $V(G_i)$ , that is now  $|B_i| \leq 1$  for each  $i$ .

If there exists  $i$  such that  $G_i$  contains at least 2 branch vertices, we modify the spanning tree so that only one remains. We proceed as follows.

While there exists  $i$  such that  $|V(G_i) \cap B| \geq 2$ .

- Choose any  $j \neq i$  such that  $\{i, j\} \in E(G)$  and let

$$w \in \begin{cases} B_j & \text{if } B_j = \{w\}, \\ V(G_j) \cap B & \text{if } B_j = \emptyset \text{ and } V(G_j) \cap B \neq \emptyset \\ V(G_j) & \text{otherwise.} \end{cases}$$

- For each branch vertex  $v \in V(G_i) \cap B$  with  $v \neq w$ , perform the following step.
  - \* Consider the path connecting  $v$  and  $w$  in  $T$ , say  $v, v', \dots, w$ , and modify  $T$  as follows: For any  $x \in N_T(v) \cap V(G_i)$  and  $x \neq v'$ , substitute in  $T$  the edge  $\{v, x\}$  by the edge  $\{w, x\}$ .  
(This is possible by (1). Moreover, even if now  $w$  becomes a new branch vertex, we know that  $|V(G_j) \cap B| = 1$ ; finally,  $|B_j| \leq 1$ ,  $|V(G_i) \cap B| \leq 1$ , and the number of branch vertices does not increase.

By iterating the above steps, one can obtain the desired spanning tree of  $H$  with at most one branch vertex in each  $V(G_i)$ . ◀

In the remaining part of this section, we present an algorithm that computes a spanning tree of  $H = G(G_1, \dots, G_n)$  with  $b(H)$  branch vertices, if  $b(H) > 0$ . In Section 4.2 we deal with the case  $b(H) = 0$ , that is, we show how to find a Hamiltonian path of  $H$ , if any exists.

By exploiting Lemma 4, the algorithm proceeds by considering all the subsets  $B_G \subseteq \{1, \dots, n\}$  with  $|B_G| \geq 1$ , ordered by size, and checking whether there exists a spanning tree of  $H$  with  $|B_G|$  branch vertices, so that exactly one branch vertex belongs to each  $V(G_i)$  with  $i \in B_G$  and none to each  $V(G_i)$  with  $i \notin B_G$ .

The identification of the spanning tree of  $H$  goes through the solution of an Integer Linear Program that uses the values  $ham(G_i)$ ,  $spi(G_i)$ ,  $|V(G_i)|$ , for  $i = 1, \dots, n$ , and exploits property (1). Namely, if the ILP does not admit a solution for  $B_G$ , then the set is discarded; if for  $B_G$  the ILP admits a solution, we will show how to use the partition of  $G_i$  given in  $\mathcal{P}_{ham(G_i)}$  and  $\mathcal{P}_{spi(G_i)}$  to construct a spanning tree of  $H$  having exactly  $|B_G|$  branch vertices (recall that the sets  $B_G$  are considered by increasing size). The optimal spanning tree will be indeed shown to correspond to the smallest  $B_G$  for which the ILP admits a solution.

### 3.1 The Integer Linear Program

Let  $B_G \subseteq \{1, \dots, n\}$ , with  $|B_G| \geq 1$ . Construct a digraph

$$G_{B_G} = (\{1, \dots, n\} \cup \{s\}, A_{B_G}),$$

where  $s \notin V(H)$  is an additional vertex that will be called the source.  $G_{B_G}$  is obtained from  $G$  by replacing each edge  $\{i, j\} \in E(G)$  by the two directed arcs  $(i, j)$  and  $(j, i)$ , and then adding a directed arc  $(s, r)$  where  $r$  is an arbitrary vertex in  $B_G$ . Formally,

$$A_{B_G} = \{(s, r)\} \cup \{(i, j), (j, i) \mid \text{there exists an edge between } i \text{ and } j \text{ in } E(G)\}.$$

For sake of clearness, we will refer to the vertices of  $G$  as *module indices* and reserve the term vertex to those in  $H$ .

We use the solution of the following Integer Linear Programming (ILP) to select arcs of  $G_{B_G}$  that will help to construct the desired spanning tree in  $H$ .

$$x_{sr} = 1 \tag{2}$$

$$\sum_{j:(j,i) \in A_{B_G}} x_{ji} \leq |V(G_i)| \quad \forall i \in \{1, \dots, n\} \tag{3}$$

$$\sum_{j:(j,i) \in A_{B_G}} x_{ji} \geq spi(G_i) \quad \forall i \in B_G \tag{4}$$

$$\sum_{j:(j,i) \in A_{B_G}} x_{ji} \geq ham(G_i) \quad \forall i \in \{1, \dots, n\} - B_G \tag{5}$$

$$\sum_{\ell:(i,\ell) \in A_{B_G}} x_{i\ell} - \sum_{j:(j,i) \in A_{B_G}} x_{ji} \leq 0 \quad \forall i \in \{1, \dots, n\} - B_G \tag{6}$$

$$y_{sr} = n \tag{7}$$

$$\sum_{j:(j,i) \in A_{B_G}} y_{ji} - \sum_{\ell:(i,\ell) \in A_{B_G}} y_{i\ell} = 1 \quad \forall i \in \{1, \dots, n\} \tag{8}$$

$$y_{ij} \leq n x_{ij} \quad \forall (i, j) \in A_{B_G} \tag{9}$$

$$y_{ij}, x_{ij} \in \mathbb{N} \quad \forall (i, j) \in A_{B_G} \tag{10}$$

For each arc  $(i, j) \in A_{B_G}$ , the non negative decision variable  $x_{ij}$  represents the load to be put on  $(i, j)$ . The load of the arc  $(s, r)$  is set to 1. The total incoming load at module index  $i \in \{1, \dots, n\}$  has to be at most  $|V(G_i)|$  and at least  $spi(G_i)$  in case  $i \in B_G$  (to be sure that the spider and all the  $spi(G_i) - 1$  paths in  $G_i$  are reached) and at least  $ham(G_i)$  in case  $i \notin B_G$  (to be sure that all the  $ham(G_i)$  paths in  $G_i$  are reached). Constraints (3), (4) and (5) correspond to this requirement.

Constraint (6) binds the relation between the total incoming and outgoing loads at any  $i \notin B_G$ , namely  $i$  must have an outgoing load upper bounded by its incoming load.

Constraints (7) and (8) use a single commodity flow in which  $s$  is used as the source and the other module indices are demand vertices. For each arc  $(i, j) \in A_{B_G}$ , the non negative decision variable  $y_{ij}$  represents the quantity of flow from  $i$  to  $j$ .

Each  $i \in \{1, \dots, n\}$  has demand of one unit; therefore, the difference between the inflow and the outflow must be exactly one. Meanwhile, the supply quantity at the source  $s$  has to be exactly  $n$ , in order to reach each of the module index in  $\{1, \dots, n\}$ .

Constraint (9) stresses variable  $y_{ij} = 0$  whenever  $x_{ij} = 0$ ; thus if no load is put on  $(i, j)$  then  $j$  cannot be reached through  $i$ .

Given an integer solution  $(y, x)$ , if any, to the above ILP, the values of variables  $y$  imply that each module index  $i \in \{1, \dots, n\} - \{r\}$  is reached from the source  $s$ . Then, by the construction of the digraph  $G_{B_G}$ , each module index  $i \in \{1, \dots, n\}$  is reached from module index  $r$ . Furthermore, by the relation between variables  $x$  and  $y$  (constraint (9)), we know that each module index  $i \in \{1, \dots, n\}$  gets incoming load from at least one of its neighbors.

▷ Claim 5. The subgraph  $G_x$  of  $G_{B_G}$  with vertex set  $\{1, \dots, n\}$  and arc set  $\{(i, j) \mid x_{ij} \geq 1\}$  contains a directed path from  $r$  to any other module index .

We stress that the constraints involving variables  $y$  only assure that a spanning tree in  $G_x$  exists. A more sophisticated approach is necessary to find a spanning tree of  $H$  with branch vertices in  $B_G$  only.

### 3.2 The spanning tree construction

Our algorithm constructs a spanning tree  $T$  of  $H$  with  $|B_G|$  branch vertices, one in each  $V(G_i)$  with  $i \in B_G$ . To this aim, it uses the values of variables  $x$  and the path-spider cover  $\mathcal{P}_{spi(G_i)}$  for  $i \in B_G$  and the partition into disjoint paths  $\mathcal{P}_{ham(G_i)}$  for  $i \notin B_G$ .

Denote by  $In(i)$  the set of the module indices for which there exist arcs in  $G_x$  toward  $i$ , that is,  $In(i) = \{j \mid x_{ji} \geq 1\}$ , and by

$$\alpha_i = \sum_{j: j \in In(i)} x_{ji} \tag{11}$$

the number of vertices of  $V(G_i)$  whose parent in  $T$  is a vertex outside  $V(G_i)$ .

Let  $\mathcal{P}^i = \{P_1^i, P_2^i, \dots, P_{\alpha_i}^i\}$  be

- the path-spider cover of  $G_i$  obtained from those in  $\mathcal{P}_{spi(G_i)}$  by removing  $\alpha_i - spi(G_i)$  arbitrary edges in case  $i \in B_G$  (notice that by constraint (4), it holds  $\alpha_i \geq spi(G_i)$ ), or
- the partition of  $G_i$  into disjoint paths obtained from those in  $\mathcal{P}_{spi(G_i)}$  by removing  $\alpha_i - ham(G_i)$  arbitrary edges in case  $i \notin B_G$  (notice that by (4), it holds  $\alpha_i \geq ham(G_i)$ ).

Furthermore, denote by

$$f(\mathcal{P}^i) = \{f(P_1^i), f(P_2^i), \dots, f(P_{\alpha_i}^i)\}$$

the sets of the first end-points in the partition  $\mathcal{P}^i$  and by

$$s(\mathcal{P}^i) = \{s(P_1^i), s(P_2^i), \dots, s(P_{\alpha_i}^i)\}$$

the sets of the second end-points in  $\mathcal{P}^i$ . In case  $i \in B_G$ , we assume that  $P_1^i \in \mathcal{P}^i$  is the spider and  $f(P_1^i) = s(P_1^i)$  is the center in  $P_1^i$ .

We also denote by

$$\beta_i = \begin{cases} \sum_{\ell: i \in I_n(\ell)} x_{i\ell} & \text{if } i \notin B_G \\ 1 & \text{if } i \in B_G \end{cases} \quad (12)$$

the number of vertices of  $V(G_i)$ , that will be the parent of some vertex in  $\bigcup_{\ell: i \in I_n(\ell)} V(G_\ell)$ .

Our algorithm ensures that the  $\alpha_i$  vertices in  $f(\mathcal{P}^i)$  are the vertices in  $G_i$  whose parent in  $T$  is outside  $V(G_i)$ , and that  $\beta_i$  vertices among those in  $s(\mathcal{P}^i)$  are chosen to be parents of vertices outside  $V(G_i)$ . Notice that by Claim 5 ( $\alpha_i \geq 1$ ) and constraint (6), it follows that  $\alpha_i \geq \beta_i$  for each  $i \in \{1, \dots, n\}$ .

Figure 2 shows the partition of graph  $G_i$  (whose vertices are grouped in the dotted circle) into  $\alpha_i$  disjoint paths if  $i \notin B_G$  and into a spider plus  $\alpha_i - 1$  disjoint paths if  $i \in B_G$ .



■ **Figure 2** The vertices of graph  $G_i$ , grouped in the dotted circle, as partitioned in  $\alpha_i$  disjoint paths if  $i \notin B_G$  and in a spider plus  $\alpha_i - 1$  disjoint paths if  $i \in B_G$ . Vertex  $f(P_j^i)$  is the only vertex in  $P_j^i$  whose parent in  $T$  is outside  $G_i$  and vertex  $s(P_j^i)$  is the only vertex in  $P_j^i$  that can have a children in  $T$  outside  $G_i$ .

The algorithm TREE constructs a spanning tree of  $H = G(G_1, \dots, G_n)$  iteratively by exploring unexplored vertices of  $H$ , until possible, and maintains a main subtree  $T$  and a forest whose roots are progressively connected to  $T$  to assemble the spanning tree. The process stops when all the vertices of  $H$  are explored. A similar idea was used in [15] in the special case in which each graph  $G_i$  is either a clique or an independent set.

The algorithm uses a queue  $Q$  to enqueue the explored vertices and maintains a set  $R$  of the roots of trees of explored vertices that wait to be connected to the main tree  $T$ . The forest structure is described through the parent function  $\pi$ .

At the beginning the set  $R$  is empty. The exploration starts with the center  $f(P_1^r)$  of the spider in the path-spider cover of  $G_r$  (recall that by construction  $r \in B_G$ ); the procedure EXPLORE( $f(P_1^r)$ ) carries out the construction of the main tree  $T$  rooted at  $f(P_1^r)$  and marks as explored all the reached vertices (adding them to the set  $Ex$ ). Clearly, for each explored vertex  $v$  there is a path in  $T$  joining  $f(P_1^r)$  to  $v$ .

However, it can occur that some of the vertices have not been explored (i.e.,  $V(H) - Ex \neq \emptyset$ ). In such a case an explored vertex  $w \in f(\mathcal{P}^j) \cap Ex$  is chosen so that it belongs to some  $V(G_j)$  which also contains at least a unexplored vertex  $u \in (f(\mathcal{P}^j) - Ex) - R$  which is able to explore at least one unexplored neighbour outside  $G_j$ , that is,  $\beta_j \geq 1$  (the existence of such a set  $V(G_j)$  is assured by Lemma 7). By using (1) and knowing that the parents of vertices in  $f(\mathcal{P}^j)$  are outside  $V(G_j)$ , the algorithm makes:

- the parent of  $w$  (recall that  $w$  is explored) become the parent of  $u$ , and
- $w$  (the root of a subtree of explored vertices) is added to  $R$  and removed from  $Ex$  (this allows to later explore  $w$  and add it, together with its subtree, to the main tree  $T$ ), and
- EXPLORE( $u$ ) is called to start a new exploration from  $u$ .

Notice that the algorithm modifies the forest by assigning to  $u$  the parent of  $w$  and only later (after adding  $u$  and some descendants of  $u$ ) adding again the subtree rooted in  $w$  to the main tree  $T$ . This allows connecting new vertices in  $V(H) - Ex$  to the main tree  $T$ ; the particular choice of  $u$  and  $w$  will be shown to avoid the possibility that the algorithm fails, due to the fact that no arc can be added to  $T$  without forming a cycle or creating an extra branch vertex. The process is iterated as long as there are unexplored vertices, i.e.  $V(H) - Ex \neq \emptyset$ .

The procedure  $\text{EXPLORE}(u)$  starts exploring  $u = f(P_k^j)$  together with the whole  $P_k^j$ \*, then putting in  $Q$  only the vertex  $s(P_k^j)$  and successively padding the main tree  $T$  (recall that (unless  $u = f(P_1^r)$ ) the parent of  $u$  is a vertex already in  $T$ , thus we construct a subtree rooted at  $u$  spanning on all the newly explored vertices).  $\text{EXPLORE}(u)$  uses for each module index  $i$  the values of  $\alpha_i$  and  $\beta_i$  that are initially defined as in (11) and (12), and the partition  $\mathcal{P}^i = \{P_1^i, P_2^i, \dots, P_{\alpha_i}^i\}$  of  $G_i$ . The value  $\alpha_i = \sum_{j: j \in \text{In}(i)} x_{ji}$  counts the number of vertices of  $V(G_i)$  that must be assigned a parent outside  $V(G_i)$ , they are the vertices in  $f(\mathcal{P}^i) = \{f(P_1^i), f(P_2^i), \dots, f(P_{\alpha_i}^i)\}$ ; in particular,  $x_{ji}$  vertices of  $f(\mathcal{P}^i)$  have to be explored by vertices in  $V(G_j)$ , for  $j \in \text{In}(i)$ . The value  $\beta_i$  counts the number of vertices of  $V(G_i)$  that have to explore other vertices in some other  $V(G_\ell)$ , for  $\ell: i \in \text{In}(\ell)$ ; in particular,

- if  $i \in B_G$  then exactly  $\beta_i = 1$  vertex in  $V(G_i)$ , that is  $s(P_1^i)$  (i.e., the center of the spider  $P_1^i$ ), becomes a branch vertex in  $T$ : it is set as the parent of  $x_{i\ell}$  unexplored vertices in  $f(\mathcal{P}^\ell)$  for each  $\ell$  such that  $x_{i\ell} \geq 1$  (i.e.,  $i \in \text{In}(\ell)$ ), and
- if  $i \notin B_G$  then  $\beta_i = \sum_{\ell: i \in \text{In}(\ell)} x_{i\ell}$  vertices in  $s(\mathcal{P}^i) = \{s(P_1^i), s(P_2^i), \dots, s(P_{\alpha_i}^i)\}$  are chosen and each one becomes the parent of one unexplored vertex in  $f(\mathcal{P}^\ell)$ .

Recall that, by the ILP constraints, we know that  $\alpha_i \geq \beta_i$ .

The vertices in  $s(\mathcal{P}^i)$  for  $i \in \{1, \dots, n\}$  are the only one to be enqueued in  $Q$ . When a vertex  $v \in s(\mathcal{P}^i)$  is dequeued from  $Q$  in  $\text{EXPLORE}(u)$  then the value of  $\beta_i$  is decreased by one if  $v$  explores (i.e., if  $\beta_i \geq 1$ ). In this case, for each explored vertex  $f(P_h^\ell)$ , with  $i \in \text{In}(\ell)$ , the whole  $P_h^\ell$  is also explored. Furthermore, the value  $\alpha_\ell$  is decreased by the number of vertices in  $s(\mathcal{P}^\ell)$  that  $v$  explores, for  $i \in \text{In}(\ell)$ . Hence, at the beginning of each iteration of the while loop in  $\text{EXPLORE}(u)$  the values of  $\alpha_i$  represents the number of vertices in  $f(\mathcal{P}^i)$  that remain to be explored while  $\beta_i$  is the number of vertices in  $s(\mathcal{P}^i)$  that already have to explore. Note that when a vertex  $v \in s(\mathcal{P}^i)$  is dequeued from  $Q$  in  $\text{EXPLORE}(u)$ , with  $u \neq f(P_1^r)$ , and  $\beta_i \geq 1$ , the algorithm checks if the neighbour  $v' \in f(\mathcal{P}^\ell)$ , that  $v$  explores, is in  $R$  (i.e.,  $v'$  is a root of a tree in the forest). In this case  $v'$ , with the tree rooted at it, is connected to the main tree  $T$  (since it was already explored in the past).

► **Lemma 6.** *At the end of  $\text{EXPLORE}(f(P_1^r))$  the function  $\pi$  describes a tree, rooted at  $f(P_1^r)$ , spanning the set  $Ex \subseteq V(H)$  of explored vertices. The vertices in  $B \cap Ex$  are the branch vertices.*

**Proof.** When  $\text{EXPLORE}(f(P_1^r))$  is called, the whole spider  $P_1^r$  is explored (i.e.  $Ex = Ex \cup P_1^r$  and so added to  $T$ ) and its center  $s(P_1^r)$  is enqueued in  $Q$ . After that, each time a vertex  $v \in s(\mathcal{P}^i)$  is dequeued from  $Q$  (recall,  $v \in Ex$ , i.e., it is an explored vertex), the algorithm can either stop its exploration (i.e.,  $\beta_i = 0$ ) or explore one or more unexplored neighbor of  $v$  together with the path/spider it belongs. Indeed, we can prove that  $v$  has the needed number of unexplored neighbors. If  $\beta_i = 0$  then  $v$  is a leaf in  $T$ ; hence, we only have to consider the case  $\beta_i \geq 1$ . If  $i \notin B_G$  then  $v$  has  $\beta_i \geq 1$  unexplored neighbors and one of them, say  $f(P_h^\ell)$

\* Assume that when in the algorithm  $P \in \mathcal{P}^j$  is explored, that is  $Ex = Ex \cup P$ , then the parent function  $\pi$  is set going through all the vertices in  $P$  from  $f(P)$  to  $s(P)$  in case  $P$  is a path, and from the center  $f(P) = s(P)$  to the leaves in case  $P$  is a spider.

■ **Algorithm 1** TREE( $H, G_1, \dots, G_n, r, B_G$ ).

---

```

1:  $R = \emptyset, B = \emptyset, Ex = \emptyset$ 
2:  $\pi(u) = \text{nhil}$  for each  $u \in V(H)$ 
3: EXPLORE( $f(P_1^r)$ )
4: while  $V(H) - Ex \neq \emptyset$  do
5:   - Let  $G_j$  be any graph s.t.  $((f(\mathcal{P}^j) - Ex) - R \neq \emptyset \neq f(\mathcal{P}^j) \cap Ex)$  and  $\beta_j \geq 1$ 
6:   - Let  $w \in f(\mathcal{P}^j) \cap Ex$  and  $u \in (f(\mathcal{P}^j) - Ex) - R$ 
7:   - Set  $\pi(u) = \pi(w), Ex = Ex - \{w\}, R = R \cup \{w\}$ 
8:   - EXPLORE( $u$ )
9: end while
10: return  $\pi, B$ 

```

---

for  $i \in In(\ell)$ , can be added to  $T$  as child of  $v$ . If  $i \in B_G$  then  $v$  is the first vertex of  $V(G_i)$  that explores and  $x_{i\ell}$  vertices in  $f(\mathcal{P}_\ell)$  are unexplored and can be added to  $T$  as children of  $v$ , for each  $\ell$  such that  $x_{i\ell} \geq 1$ . Hence  $v$  becomes a branch vertex in  $T$  and is put in  $B$ . Since  $R = \emptyset$  (i.e. no tree is in the forest), each time a neighbor of  $v$  is explored, say  $f(P_h^\ell)$ , then the whole path/spider  $P_h^\ell$  is added to  $T$  and  $s(P_h^\ell)$  is enqueued in  $Q$ . Hence, any explored vertex has  $f(P_1^r)$  as ancestor, i.e., the function  $\pi$  describes a path joining any explored vertex to  $f(P_1^r)$ . Noticing that no vertex can be enqueued twice in  $Q$  (since any enqueued vertex is also marked as explored), we have that the function  $\pi$  does not create cycles. ◀

We can also prove the following result.

► **Lemma 7.** *Let  $Ex$  be the set of explored vertices at the beginning of any iteration of the while loop in algorithm TREE. If  $V(H) - Ex \neq \emptyset$  then there exists a module index  $j$  such that  $(f(\mathcal{P}^j) - Ex) - R \neq \emptyset \neq f(\mathcal{P}^j) \cap Ex$  and  $\beta_j \geq 1$ .*

► **Lemma 8.** *After each call of EXPLORE( $u$ ) the function  $\pi$  describes a forest spanning the vertices in  $Ex \cup R$  of explored vertices and consisting of  $|R| + 1$  trees respectively rooted at  $f(P_1^r)$  and at the vertices in  $R$ . The vertices in  $B$  are the only branch vertices in the forest.*

**Proof.** When EXPLORE( $u$ ) is called, the function  $\pi$  describes a forest, spanning the current set  $Ex \cup R$ , whose roots are the vertices in  $\{f(P_1^r)\} \cup R$  and where  $R \subset V(H) - Ex$ . We notice that by Lemma 6, this is true the first time EXPLORE is called, that is, after the call to EXPLORE( $f(P_1^r)$ ) (at that time  $R = \emptyset$ ).

We prove that the claim is also true at the end of each call to EXPLORE( $u$ ). When EXPLORE( $u$ ) is called,  $Q$  is empty; vertex  $u$  is explored (i.e. it is added to  $Ex$ ) and enqueued in  $Q$ . Then EXPLORE( $u$ ) proceeds, exactly as in EXPLORE( $f(P_1^r)$ ), dequeuing vertices from  $Q$  and exploring their unexplored neighbors, so constructing a subtree of the main tree  $T$  rooted at  $u$  described by function  $\pi$ . The only difference with EXPLORE( $f(P_1^r)$ ) is when one of the vertices explored is  $v' \in R$ . Vertex  $v' \in R$  is removed from  $R$  (see lines 11, 22) and connected to the main tree  $T$  through the function  $\pi$  and marked as explored exactly as any other explored vertex. However  $v'$  is not enqueued in  $Q$  since it has already explored its neighbors; hence,  $v'$  is connected to  $T$  together with its subtree of explored vertices. ◀

We are now able to prove the following result.

► **Lemma 9.** *The algorithm TREE returns a spanning tree of  $H$ , described by function  $\pi$ , with branch vertex set  $B$ .*

**Proof.** By using Lemma 6 we know that algorithm TREE constructs, through procedure EXPLORE( $f(P_1^r)$ ), a main tree  $T$ , described by  $\pi$ . In case  $T$  does not span all the vertices in  $V(H)$  then, Lemma 7 assures that the algorithm finds a graph  $G_j$  with an explored vertex

---

**Algorithm 2** EXPLORE( $u$ ).
 

---

```

1: Let  $Q$  be an empty queue
2: Let  $u = f(P_k^j)$ 
3:  $Ex = Ex \cup P_k^j$ 
4:  $Q.enqueue(s(P_k^j))$ 
5: while  $Q \neq \emptyset$  do
6:    $v = Q.dequeue$ 
7:   Let  $v \in s(\mathcal{P}^i)$ 
8:   if  $i \notin B_G$  and  $\beta_i \geq 1$  then
9:     - Let  $f(P_h^\ell) \in f(\mathcal{P}^\ell) - Ex$  for some  $\ell$  s.t.  $i \in In(\ell)$ 
10:    -  $\pi(f(P_h^\ell)) = v$ 
11:    if  $f(P_h^\ell) \notin R$  then
12:      -  $Ex = Ex \cup P_h^\ell$ 
13:      -  $Q.enqueue(s(P_h^\ell))$ 
14:    else  $R = R - \{f(P_h^\ell)\}$ 
15:    end if
16:    -  $\alpha_\ell = \alpha_\ell - 1$ 
17:    -  $\beta_i = \beta_i - 1$ 
18:  else if  $i \in B_G$  and  $\beta_i = 1$  then
19:    -  $B = B \cup \{v\}$ 
20:    for each  $\ell$  s.t.  $i \in In(\ell)$  do
21:      - Let  $A_{i\ell} \subseteq f(\mathcal{P}^\ell) - Ex$  s.t.  $|A_{i\ell}| = x_{i\ell}$ 
22:      -  $\alpha_\ell = \alpha_\ell - x_{i\ell}$ ,
23:      for each  $f(P_h^\ell) \in A_{i\ell}$  do
24:        -  $\pi(f(P_h^\ell)) = v$ 
25:        if  $f(P_h^\ell) \notin R$  then
26:          -  $Ex = Ex \cup P_h^\ell$ 
27:          -  $Q.enqueue(s(P_h^\ell))$ 
28:        else  $R = R - \{f(P_h^\ell)\}$ 
29:        end if
30:      end for
31:    end for
32:    -  $\beta_i = \beta_i - 1$ 
33:  end if
34: end while

```

---

$w \in f(\mathcal{P}^j) \cap Ex$  and an unexplored vertex  $u \in (f(\mathcal{P}^j) - Ex) - R$ . Disconnecting  $w$  (together with its subtree) from the main tree  $T$ , the algorithm let  $w$  become one of the roots of trees in  $R$ . Furthermore, since the parent of  $w$  in  $T$  is a vertex outside  $V(G_j)$  and, since  $u$  and  $w$  share the same neighborhood outside  $G_j$  (by (1)), the algorithm let  $u$  be connected to the vertex that was the parent of  $w$  in  $T$  (thus, connecting  $u$  to  $T$ ). Considering that  $u \notin R$  and  $\beta_j \geq 1$ , the algorithm starts a new exploration from  $u$  (recall that  $u \in f(\mathcal{P}^j) - Ex$ ) calling EXPLORE( $u$ ). By Lemma 8, this allows padding  $T$  with the subtree rooted at  $u$ . The lemma follows by iterating the above procedure until no unexplored vertex exists in  $V(H)$ . ◀

### 3.3 The algorithm complexity

Summarizing, given the triple  $(\text{ham}(G_i), \text{spi}(G_i), |V(G_i)|)$  together with  $\mathcal{P}_{\text{ham}(G_i)}$  and  $\mathcal{P}_{\text{spi}(G_i)}$ , for each  $i \in \{1, \dots, n\}$ , the proposed method to construct the spanning tree of  $H = G(G_1, \dots, G_n)$  works as follows.

For each  $B_G \subseteq \{1, \dots, n\}$  with  $|B_G| \geq 1$ , selected in order of increasing size, the algorithm executes the following steps:

- solve the corresponding ILP
- if a solution exists for the current set  $B_G$ , use algorithm TREE to construct a spanning tree of  $H = G(G_1, \dots, G_n)$  with  $|B_G|$  branch vertices.

Jansen and Rohwedder [17] have recently showed that the time needed to find a feasible solution of an ILP with  $p$  integer variables and  $q$  constraints is  $O(\sqrt{q}\Delta)^{(1+o(1))q} + O(qp)$ , where  $\Delta$  is the largest absolute value of any coefficient in the ILP. Denoted by  $m$  the number of edges of  $G$ , our ILP has  $q = 3n + 2m + 1$  constraints,  $p = 2(m + 1)$  variables and  $\Delta = n$ . Hence the time to solve it is  $O(n\sqrt{n+m})^{(1+o(1))(3n+2m+1)} + O(n(n+m))$ . Using the solution  $(y, x)$  of the ILP, the algorithm TREE returns the spanning tree of  $H$  in time  $O(|V(H)|^2)$ . Overall, the algorithm requires time

$$2^n [O(n\sqrt{n+m})^{(1+o(1))(3n+2m+1)} + O(n(n+m))] + O(|V(H)|^2).$$

Recall that  $n \leq mw$ , and therefore  $m \leq nw^2$ .

### 3.4 Optimality

It is possible to show that if no set  $B_G \subseteq \{1, \dots, n\}$ , of size  $k$  exists for which the ILP admits a solution then any spanning tree of  $H = G(G_1, \dots, G_n)$  has  $b(H) \geq k + 1$  branch vertices. This allows to say that the optimal spanning tree in  $H$  corresponds to the smallest set  $B_G \subseteq \{1, \dots, n\}$  for which the ILP admits a solution, if any. Namely, we can prove the following result.

► **Lemma 10.** *Given the graphs  $G_1, \dots, G_n$  and  $\text{ham}(G_i), \text{spi}(G_i), |V(G_i)|$  for each  $i = 1, \dots, n$ , if there exists a spanning tree in  $H = G(G_1, \dots, G_n)$  with  $k \geq 1$  branch vertices then there exist a set  $B_G \subseteq \{1, \dots, n\}$  with  $|B_G| = k$ , for which ILP admits a solution  $(x, y)$ .*

## 4 The triple and partition computation

In this section we show how to compute the record of data for any vertex  $\hat{H} = \hat{G}(\hat{G}_1, \dots, \hat{G}_{\hat{n}})$  of the parse-tree. Namely, given the triple  $(\text{ham}(\hat{G}_i), \text{spi}(\hat{G}_i), |V(\hat{G}_i)|)$ ,  $\mathcal{P}_{\text{ham}(\hat{G}_i)}$  and  $\mathcal{P}_{\text{spi}(\hat{G}_i)}$ , for each  $i = 1, \dots, \hat{n}$ , we have to compute

the triple  $(\text{ham}(\hat{H}), \text{spi}(\hat{H}), |V(\hat{H})|)$  together with  $\mathcal{P}_{\text{ham}(\hat{H})}$  and  $\mathcal{P}_{\text{spi}(\hat{H})}$ ,

Clearly,  $|V(\hat{H})| = \sum_{i=1}^{\hat{n}} |V(\hat{G}_i)|$ . We show below how to compute  $\text{spi}(\hat{H})$  and  $\mathcal{P}_{\text{spi}(\hat{H})}$ , and also  $\text{ham}(\hat{H})$  and  $\mathcal{P}_{\text{ham}(\hat{H})}$ ,

For a graph  $\hat{H}$  and an integer  $\ell$ , we denote by  $\hat{H} \otimes \ell$  the graph obtained from  $\hat{H}$  by adding  $\ell$  vertices and connecting them to every vertex in  $\hat{H}$ ; formally,  $\hat{H} \otimes \ell$  has vertex set  $V(\hat{H}) \cup \{v_1, \dots, v_\ell\}$  and edge set  $E(\hat{H}) \cup \{\{u, v_j\} \mid u \in V(\hat{H}), 1 \leq j \leq \ell\}$ . We notice that, since  $\hat{H} = \hat{G}(\hat{G}_1, \dots, \hat{G}_{\hat{n}})$  then the graph  $\hat{H} \otimes \ell$ , for each  $2 \leq \ell \leq |V(\hat{H})|$ , is equal to the graph  $\hat{G}'(\hat{G}_1, \dots, \hat{G}_{\hat{n}}, I_\ell)$  where  $\hat{G}'$  is the graph obtained from  $\hat{G}$  by adding the vertex  $\hat{n} + 1$  (i.e.,  $V(\hat{G}') = \{1, \dots, \hat{n}, \hat{n} + 1\}$ ) and making it adjacent to all the other vertices of  $\hat{G}$  (i.e.,  $E(\hat{G}') = \{(i, \hat{n} + 1) \mid 1 \leq i \leq \hat{n}\}$ ), and  $I_\ell$  is the independent set with  $\ell$  vertices  $\{v_1, \dots, v_\ell\}$ .

#### 4.1 Computing $\text{spi}(\hat{H})$ and $\mathcal{P}_{\text{spi}(\hat{H})}$

In order to compute the values  $\text{spi}(\hat{H})$  and  $\mathcal{P}_{\text{spi}(\hat{H})}$ , we first need a preliminary result.

► **Proposition 11.** *Let  $\hat{H}$  be a graph and*

$$s(\hat{H}) = \min\{\ell \mid \hat{H} \otimes (\ell - 1) \text{ has a spanning spider with center in } \hat{H}\}.$$

Then  $\text{spi}(\hat{H}) = s(\hat{H})$ .

**Proof.** We first show that  $s(\hat{H}) \leq \text{spi}(\hat{H})$ . Let  $P_1, P_2, \dots, P_{\text{spi}(\hat{H})}$  be the path-spider cover of  $\hat{H}$  and let  $f(P_1)$  be the center of spider  $P_1$  and,  $f(P_i)$  be the first end-point of path  $P_i$  for  $i = 2, \dots, \text{spi}(\hat{H})$ . Hence, the graph  $\hat{H} \otimes (\text{spi}(\hat{H}) - 1)$  contains the spider with center  $f(P_1)$  obtained connecting  $f(P_1)$  to the vertex  $i$  and then connecting vertex  $i$  to  $f(P_i)$  for each for  $i = 1, \dots, \text{spi}(\hat{H}) - 1$ . Now, we prove that  $\text{spi}(\hat{H}) \leq s(\hat{H})$ . Let  $S$  be a spider in  $\hat{H} \otimes (s(\hat{H}) - 1)$  with the center  $u \in V(\hat{H})$ . Then, removing from  $S$  the vertices in  $\{1, \dots, s(\hat{H}) - 1\}$  we have a path-spider cover with center  $u$  and  $s(\hat{H}) - 1$  path pairwise disjoint. ◀

Recall that the graph  $\hat{H} \otimes (\ell - 1)$  is equal to  $\hat{G}'(\hat{G}_1, \dots, \hat{G}_n, I_{\ell-1})$  and notice that

$$\text{ham}(I_{\ell-1}) = \text{spi}(I_{\ell-1}) = \ell - 1 \text{ and } \mathcal{P}_{\text{ham}(I_{\ell-1})} = \mathcal{P}_{\text{spi}(I_{\ell-1})} = I_{\ell-1}.$$

We can then take into account the values  $\text{ham}(\hat{G}_i)$ ,  $\text{spi}(\hat{G}_i)$ ,  $|V(\hat{G}_i)|$ , and the sets  $\mathcal{P}_{\text{ham}(\hat{G}_i)}$  and  $\mathcal{P}_{\text{spi}(\hat{G}_i)}$ , for all  $i \in \{1, \dots, \hat{n}\}$ . For each  $B_{\hat{G}_i} = \{j\}$ , for  $j = 1, \dots, \hat{n}$ , we can follow the lines of Section 3.1 to verify whether the corresponding ILP is feasible. In the positive case, following the construction given in Section 3.2, one can obtain a spider of  $\hat{H} \otimes (\ell - 1)$  centered in  $V(\hat{G}_i)$

The minimum  $\ell$  for which the above occurs, gives  $\text{spi}(\hat{H})$  as well as the spider  $T$  covering  $\hat{H} \otimes (\text{spi}(\hat{H}) - 1)$  with center in  $V(\hat{H})$ . The arguments used in Section 2.3 allows to obtain the time complexity of this computation (here,  $\hat{m}$  represents the number of edges of  $\hat{G}$ )

$$\text{spi}(\hat{H}) \hat{n} [O(\hat{n}\sqrt{\hat{n} + \hat{m}})^{(1+o(1))(3\hat{n}+2\hat{m}+1)} + O(\hat{n}(\hat{n} + \hat{m}))] + O(|V(\hat{H})|^2).$$

Clearly, the subgraph of  $T$  induced by  $V(\hat{H})$  returns the path-spider cover  $\mathcal{P}_{\text{spi}(\hat{H})}$  of  $\hat{H}$ , thus concluding the proof of Theorem 3.

#### 4.2 Computing $\text{ham}(\hat{H})$ and $\mathcal{P}_{\text{ham}(\hat{H})}$

Using an approach similar to the one in the proof of Proposition 11, we can prove the following result.

► **Proposition 12.** *Let  $\hat{H}$  be a graph and*

$$h(\hat{H}) = \min\{\ell \mid \hat{H} \otimes \ell \text{ has a hamiltonian path with an end-point in } \{v_1, \dots, v_\ell\}\}.$$

Then  $\text{ham}(\hat{H}) = h(\hat{H})$ .

By Proposition 12, the value  $\text{ham}(\hat{H})$  is equal to the minimum positive integer  $\ell$  with  $1 \leq \ell \leq |V(\hat{H})|$  such that the graph

$$\hat{H} \otimes \ell \text{ has a hamiltonian path with an end-point not in } \{v_1, \dots, v_\ell\}. \quad (13)$$

To verify if graph  $\hat{H} \otimes \ell$  has a hamiltonian path and eventually find it, we can proceed as in Section 3. Indeed, considering that  $\hat{H} \otimes \ell = \hat{G}'(\hat{G}_1, \dots, \hat{G}_n, I_\ell)$ , and that  $\text{ham}(I_\ell) = \ell$  and  $\mathcal{P}_{\text{ham}(I_\ell)} = I_\ell$ , then given the values  $\text{ham}(\hat{G}_i)$ ,  $|V(\hat{G}_i)|$  and the set  $\mathcal{P}_{\text{ham}(\hat{G}_i)}$ , for each

$i \in \{1, \dots, \hat{n}\}$ , we can consider the corresponding ILP as in Section 3.1 choosing  $B_{\hat{G}_r} = \emptyset$  and  $r = \hat{n} + 1$  (i.e.,  $\hat{G}_r = I_\ell$ ). If the ILP admits a solution, we can construct the hamiltonian path  $P$  of  $\hat{H} \otimes \ell = \hat{G}'(\hat{G}_1, \dots, \hat{G}_{\hat{n}}, I_\ell)$  following the construction in Section 3.2 choosing any vertex in  $\hat{G}_r = I_\ell$  as end-point (i.e., root). Finally, we notice that everything was proved in Section 3.2 holds also in this case (i.e.,  $\hat{G}'(\hat{G}_1, \dots, \hat{G}_{\hat{n}}, I_\ell)$ ,  $B_{\hat{G}_r} = \emptyset$  and  $\hat{G}_r = I_\ell$ ) and that, as in Section 3.4, it can be proved that if there exists a hamiltonian path of  $\hat{G}'(\hat{G}_1, \dots, \hat{G}_{\hat{n}}, I_\ell)$  with an end-point in  $I_\ell$  then exists a solution  $(x, y)$  of the corresponding ILP (the same arguments used in the proof of Lemma 10 holds rooting the hamiltonian path at the end-point in  $I_\ell$ ).

The minimum  $\ell$  for which (13) occurs, gives  $ham(\hat{H})$  and also the hamiltonian path  $P$  of  $\hat{H} \otimes ham(\hat{H})$  with one end-point in  $I_{ham(\hat{H})}$ . The arguments used in Section 2.3 allows to have the time complexity of this computation

$$ham(\hat{H}) [O(\hat{n}\sqrt{\hat{n} + \hat{m}})^{(1+o(1))(3\hat{n}+2\hat{m}+1)} + O(\hat{n}(\hat{n} + \hat{m}))] + O(|V(\hat{H})|^2).$$

Obviously, the subgraph of  $P$  induced by  $V(\hat{H})$  will return the partition in  $ham(\hat{H})$  disjoint paths of  $\hat{H}$ ,  $\mathcal{P}_{ham(\hat{H})}$ .

We stress that  $ham(\hat{H}) = 1$  iff the graph  $\hat{H}$  has a hamiltonian path.

---

## References

- 1 J. Baste and D. Watel. An fpt algorithm for node-disjoint subtrees problems parameterized by treewidth. *The Computer Journal*, 2022.
- 2 J.-C. Bermond, L. Gargano, S. Perennes, A.A. Rescigno, and U. Vaccaro. Optimal time data gathering in wireless networks with multidirectional antennas. *Theoretical Computer Science*, 509:122–139, 2013.
- 3 J.-C. Bermond, L. Gargano, and A.A. Rescigno. Gathering with minimum delay in tree sensor networks. In *Proceedings of 15th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2008) Alexander A. Shvartsman and Pascal Felber Eds., Lecture Notes in Computer Science*, volume 5058, pages 262–276, 2008.
- 4 J.-C. Bermond, L. Gargano, and A.A. Rescigno. Gathering with minimum completion time in sensor tree networks. *Journal of Interconnection Networks*, 11:1–33, 2010.
- 5 F. Carrabs, R. Cerulli, M. Gaudio, and M. Gentili. Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications*, 56(2):405–438, 2013.
- 6 C. Cerrone, R. Cerulli, and A. Raiconi. Relations, models and a memetic approach for three degree-dependent spanning tree problems. *European Journal of Operational Research*, 232(3):442–453, 2014.
- 7 R. Cerulli, M. Gentili, and A. Iossa. Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42(3):353–370, 2009.
- 8 V. M. Chimani and J. Spoerhase. Approximating spanning trees with few branches. *Theory of Computing Systems*, 56(1):181–196, 2015.
- 9 F.V. Fomin, P.A. Golovach, D. Lokshtanov, and S. Saurabh. Clique-width: on the price of general. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Claire Mathieu Ed., pages 825–834, 2009.
- 10 J. Gajarský, M. Lampis, and S. Ordyniak. Parameterized algorithms for modular-width. In *Proceedings of 8th International Symposium on Parameterized and Exact Computation (IPEC 2013)*, Gregory Gutin and Stefan Szeider Eds., *Lecture Notes in Computer Science*, volume 8246, pages 163–176, 2013.
- 11 L. Gargano, M. Hammar, P. Hell, L. Stacho, and U. Vaccaro. Spanning spiders and light splitting switches. *Discrete Mathematics*, 285(1):83–95, 2004.

- 12 L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. In *Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, Ricardo Conejo, Eds., *Lecture Notes in Computer Science*, volume 2380, pages 355–365, 2002.
- 13 L. Gargano and A. A. Rescigno. Collision-free path coloring with application to minimum-delay gathering in sensor networks. *Discrete Applied Mathematics*, 157(8):1858–1872, 2009.
- 14 L. Gargano and A. A. Rescigno. Complexity of conflict-free colorings of graphs. *Theoretical Computer Science*, 566:39–49, 2015.
- 15 L. Gargano and A. A. Rescigno. Spanning trees with few branch vertices in graphs of bounded neighborhood diversity. In *Proceedings of 2023 International Colloquium on Structural Information and Communication Complexity (SIROCCO 2023)*, Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, Dennis Olivetti Eds., *Lecture Notes in Computer Science*, volume 13892, pages 502–519, 2023.
- 16 D. Gozuppek, S. Buhari, and F. Alagoz. A spectrum switching delay-aware scheduling algorithm for centralized cognitive radio networks. *IEEE Transactions on Mobile Computing*, 12:1270–1280, 2013.
- 17 K. Jansen and L. Rohwedderb. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, pages 1–15, 2023.
- 18 M. Landete, A. Marín, and J. L. Sainz-Pardo. Decomposition methods based on articulation vertices for degree-dependent spanning tree problems. *Computational Optimization and Applications*, 68:749–773, 2017.
- 19 A. Marin. Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. *European Journal of Operational Research*, 245(3):680–689, 2015.
- 20 H. Matsuda, K. Ozeki, and T. Yamashita. Spanning trees with a bounded number of branch vertices in a claw-free graph. *Graphs and Combinatorics*, 30:429–437, 2014.
- 21 R. A. Melo, P. Samer, and S. Urrutia. An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. *Computational Optimization and Applications*, 65(3):821–844, 2016.
- 22 A. Rossi, A. Singh, and S. Shyam. Cutting-plane-based algorithms for two branch vertices related spanning tree problems. *Optimization and Engineering*, 15:855–887, 2014.
- 23 G. Salamon. Spanning tree optimization problems with degree-based objective functions. In *Proceedings of 4th Japanese–Hungarian Sym. on Discrete Math. and Its Applications*, pages 309–315, 2005.
- 24 N. Shami and M. Rasti. A joint multi-channel assignment and power control scheme for energy efficiency in cognitive radio networks. In *Proceedings of 2016 IEEE Wireless Communications and Networking Conference (WCNC 2016)*, pages 1–6, 2016.
- 25 R. M. A. Silva, D. M. Silva, M. G. C. Resende, G. R. Mateus, J. F. Gonçalves, and P. Festa. An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters*, 8(4):1225–1243, 2014.
- 26 S. Silvestri, G. Laporte, and R. Cerulli. A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. *Comput. Oper. Res.*, 81:322–332, 2017.
- 27 S. Sundar, A. Singh, and A. Rossi. New heuristics for two bounded-degree spanning tree problems. *Information Sciences*, 195:226–240, 2012.
- 28 M. Tedder, D.G. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Proceedings of 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, Igor Walukiewicz Eds., *Lecture Notes in Computer Science*, volume 5125, pages 634–645, 2008.



# Depth-3 Circuits for Inner Product

Mika Göös 

EPFL, Lausanne, Switzerland

Ziyi Guan 

EPFL, Lausanne, Switzerland

Tiberiu Mosnoi 

EPFL, Lausanne, Switzerland

---

## Abstract

---

What is the  $\Sigma_3^2$ -circuit complexity (depth 3, bottom-fanin 2) of the  $2n$ -bit inner product function? The complexity is known to be exponential  $2^{\alpha n}$  for some  $\alpha_n = \Omega(1)$ . We show that the limiting constant  $\alpha := \limsup \alpha_n$  satisfies

$$0.847\dots \leq \alpha \leq 0.965\dots$$

Determining  $\alpha$  is one of the seemingly-simplest open problems about depth-3 circuits. The question was recently raised by Golovnev, Kulikov, and Williams (ITCS 2021) and Frankl, Gryaznov, and Talebanfard (ITCS 2022), who observed that  $\alpha \in [0.5, 1]$ . To obtain our improved bounds, we analyse a covering LP that captures the  $\Sigma_3^2$ -complexity up to polynomial factors. In particular, our lower bound is proved by constructing a feasible solution to the dual LP.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Circuit complexity

**Keywords and phrases** Circuit complexity, inner product

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.51

**Funding** This work was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00026.

**Acknowledgements** We thank the anonymous reviewers for a careful reading of the paper and comments that helped us improve the presentation.

## 1 Introduction

A  $\Sigma_3$ -circuit is an unbounded-fanin depth-3 boolean circuit with an  $\vee$ -gate at the top. That is, the circuit computes an OR of CNFs. A foremost open problem in circuit complexity is to prove a lower bound of  $2^{\omega(\sqrt{n})}$  on the  $\Sigma_3$ -circuit complexity of an explicit  $n$ -bit boolean function. Current techniques can prove at best a bound of  $2^{\Omega(\sqrt{n})}$  [7, §11].

For the more restricted class of  $\Sigma_3^k$ -circuits that have fanin  $k$  at the bottom (that is, ORs of  $k$ -CNFs), we can hope for improved bounds. For example, the famous satisfiability coding lemma [14] implies that the  $n$ -bit parity function has  $\Sigma_3^k$ -circuit complexity at least  $2^{n/k}$  and this is tight up to polynomial factors (for constant  $k$ ). Even stronger, for  $k = 2$ , Paturi, Saks, and Zane [12] exhibit a function with near-maximal  $\Sigma_3^2$ -complexity  $2^{n-o(n)}$ . No such near-maximal lower bounds are currently known for  $k = 3$ .

**Inner product.** A natural function whose  $\Sigma_3^k$ -complexity remains unknown (up to  $\text{poly}(n)$  factors) is the *inner product* function  $\text{IP}_n$ , defined on  $2n$ -bit inputs  $(x, y) \in (\{0, 1\}^n)^2$  by

$$\text{IP}_n(x, y) := \langle x, y \rangle \bmod 2.$$

Recently, Golovnev, Kulikov, and Williams [2] asked to determine the  $\Sigma_3^k$ -complexity of  $\text{IP}_n$  in case  $k = 3$ . Curiously enough, Frankl, Gryaznov, and Talebanfard [1] point out that the problem is nontrivial already in case  $k = 2$ , and they obtained partial results towards resolving it. It has been known that the  $\Sigma_3^2$ -complexity of  $\text{IP}_n$  is between  $2^{n/2}$  and  $2^n$  [14, 2].



© Mika Göös, Ziyi Guan, and Tiberiu Mosnoi;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 51; pp. 51:1–51:12

Leibniz International Proceedings in Informatics



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

## 1.1 Our result

Our main result is to prove improved upper and lower bounds for inner product.

► **Theorem 1** (Main result). *Write the  $\Sigma_3^2$ -complexity of  $\text{IP}_n$  as  $2^{\alpha n}$  for some  $\alpha_n \geq 0$ . Then*

$$\alpha := \limsup \alpha_n \in [0.847\dots, 0.965\dots].$$

It remains an intriguing problem to determine  $\alpha$  precisely. It is surprising (for us, at least) that neither of the previous bounds  $\alpha \in [0.5, 1]$  were tight, especially because the problem is seemingly one of the simplest open questions about depth-3 circuits.

Studying exact exponents of  $\Sigma_3^k$ -circuit complexities is a relatively unexplored research direction, and we believe it could foster the development of new lower bound techniques. In particular, a major motivation for this comes from *depth reduction* results. For example, in case  $k = 16$ , Golovnev, Kulikov, and Williams [2] have shown that proving near-maximal  $2^{n-o(n)}$  bounds for  $\Sigma_3^{16}$ -circuits would already yield new improved lower bounds for *unrestricted* (unbounded depth) circuits. Their result extends classical connections discovered by Valiant [15]; see also the monograph [16, §3].

## 1.2 Overview of techniques

To obtain our improved bounds on  $\alpha$  in Theorem 1 – both upper and lower bounds – we study a fractional covering problem, formulated as a linear program (LP), that captures the  $\Sigma_3^2$ -circuit complexity up to  $\text{poly}(n)$  factors.

To our knowledge, LPs have not been widely employed in analysing depth-3 circuits. They are, however, routinely used to prove strong lower bounds in the related area of *communication complexity* [9]. Many such LP-based methods are catalogued by Jain and Klauck [6]. Moreover, Lee and Shraibman [10] give a monograph-length treatment on how to use LP duality to prove communication lower bounds. In one of the earliest examples, Karchmer, Kushilevitz, and Nisan [8] characterised nondeterministic communication complexity via a fractional covering problem. The formulation we use is a straightforward adaptation of this for depth-3 circuits. A similar formulation also appeared in the work of Hirahara [4] that connects depth-3 complexity with one-sided CNF approximations.

**Covering LP.** The size of a  $\Sigma_3^2$ -circuit is determined (up to  $O(n^2)$  factors) by the fanin of the top  $\vee$ -gate. Suppose a circuit with top-fanin  $m$  computes a function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$ . We can view the circuit as expressing the set of 1-inputs  $f^{-1}(1)$  as a union of  $m$  sets,

$$f^{-1}(1) = \bigcup_{i \in [m]} \phi_i^{-1}(1), \tag{1}$$

where each  $\phi_i^{-1}(1)$  is the set of inputs accepted by a 2-CNF formula  $\phi_i$ . The least top-fanin needed to compute  $f$  is then captured by the optimal *integer solutions* to the following covering LP. In this LP, we assign a fractional weight  $w_\phi \in [0, 1]$  for each 2-CNF  $\phi$  that is *consistent* with  $f$ , meaning that  $\phi(x) \leq f(x)$  for every input  $x \in \{0, 1\}^n$ . We let  $\Phi$  denote the set of all 2-CNFs consistent with  $f$ .

$$\begin{array}{ll} \min & \sum_{\phi \in \Phi} w_\phi \\ \text{subject to} & \sum_{\phi \in \Phi} w_\phi \phi(x) \geq 1, \quad \forall x \in f^{-1}(1) \\ & w_\phi \in [0, 1], \quad \forall \phi \in \Phi \end{array} \tag{LP}$$

A classic result of Lovász [11] says that the integrality gap of a covering LP is small.

► **Lemma 2** (Lovász [11]). *Let  $\text{Opt}$  and  $\text{Opt}^{\mathbb{Z}}$  denote the value of (LP) optimised over fractional solutions ( $w_\phi \in [0, 1]$ ) and integral solutions ( $w_\phi \in \{0, 1\}$ ), respectively. Then*

$$\text{Opt} \leq \text{Opt}^{\mathbb{Z}} \leq O(n) \cdot \text{Opt}.$$

Consequently, to determine the  $\Sigma_3^2$ -complexity of  $f = \text{IP}_n$  we only need to solve the fractional (LP). We will use the (LP) in Section 2 to construct circuits for  $\text{IP}_n$  that witness the upper bound  $\alpha \leq 0.965\dots$

**Dual LP.** A common method to prove a depth-3 lower bound is to estimate the number of accepting inputs for any consistent CNF, say, by  $\max_{\phi \in \Phi} |\phi^{-1}(1)| \leq C$ , and then conclude that the top-fanin must be at least  $|f^{-1}(1)|/C$ . Such arguments are standard in the *top-down* circuit lower bound literature [3, 14, 12, 13, 5].

An important generalisation of this method is to first choose a hard distribution  $\mathcal{D}$  over the 1-inputs  $f^{-1}(1)$  and then measure the size of  $\phi^{-1}(1)$  relative to  $\mathcal{D}$ . If we can show  $\max_{\phi \in \Phi} \Pr_{x \sim \mathcal{D}}[\phi(x) = 1] \leq p$ , then the top-fanin must be at least  $1/p$ . Indeed, the following optimisation problem captures the best lower bound provable with this method.

$$\begin{array}{ll} \max & 1/p \\ \text{subject to} & \sum_{x \in f^{-1}(1)} \mathcal{D}(x) \phi(x) \leq p, \quad \forall \phi \in \Phi \\ & \sum_{x \in f^{-1}(1)} \mathcal{D}(x) = 1, \\ & \mathcal{D}(x) \in [0, 1], \quad \forall x \in f^{-1}(1) \end{array} \quad (\text{Dual LP})$$

This program is not written in standard LP format as we are seemingly optimising a nonlinear function. However, it is equivalent<sup>1</sup> to  $\max \sum_x A(x)$  s.t.  $\sum_x A(x) \phi(x) \leq 1$  and  $A(x) \geq 0$ , which is the canonical dual to (LP). Hence, by strong duality, we can always prove a tight lower bound (up to polynomial factors) on depth-3 complexity by finding the right hard distribution  $\mathcal{D}$ .

**Hard distribution for IP.** What hard distribution  $\mathcal{D}$  should we choose to prove a strong lower bound for  $\text{IP}_n$ ? If we choose  $\mathcal{D}$  to be the uniform distribution over  $\text{IP}_n^{-1}(1)$ , then prior work [1, Thm 28] showed that this only yields the bound  $\alpha \geq \log \frac{4}{3} = 0.415\dots$  If we choose  $\mathcal{D}$  by sampling a pair  $(x, 1^n)$  where  $x$  is uniform random in  $\{0, 1\}^n$ , then we have effectively reduced  $\text{IP}_n$  to  $n$ -bit parity and we obtain  $\alpha \geq 0.5$  [2], which is tight for parity.

To get our improved lower bound on  $\alpha$ , we analyse a more general distribution.

**(Section 3)** We consider a distribution where the  $2n$  input bits are *iid*, that is,  $\mathcal{D}$  is the binomial distribution with some parameter  $p \in (0, 1)$ . (Note that while  $\mathcal{D}$  is not supported on  $\text{IP}_n^{-1}(1)$  it does place a constant probability mass on it.) We prove a structure lemma for consistent 2-CNFs and characterise those that have the highest acceptance probability under  $\mathcal{D}$ . Optimising the choice of  $p$ , we will obtain  $\alpha \geq \log \frac{9}{5} = 0.847\dots$

<sup>1</sup> If  $\mathcal{D}, p$  is feasible for (Dual LP), then  $A(x) := \mathcal{D}(x)/p$  is feasible and has the same objective function value in the other program. In the other direction, set  $p := 1/\sum_y A(y)$  and  $\mathcal{D}(x) := p \cdot A(x)$ .

### 1.3 Discussion and open problems

The challenge in proving a better lower bound in Theorem 1 is that our techniques rely heavily on the hard distribution having independence between the  $n$  coordinates. One way we could try to improve the lower bound is to consider a slightly more general *coordinate-wise iid* distribution. That is, we choose a distribution  $\mu$  over one coordinate pair  $(x_i, y_i) \in \{0, 1\}^2$  and then define a product distribution by  $\mathcal{D} := \mu^n := \mu \times \cdots \times \mu$ . We carried out this approach (using computer-aided calculations) only to find out that we get no improvement this way: the hardest  $\mathcal{D}$  is still the bit-wise *iid* that we consider in Section 3. A candidate for the absolute hardest distribution (not necessarily coordinate-wise *iid*) is merely a *symmetric* distribution that is invariant under permuting the  $n$  coordinates. We leave it as an open problem to analyse such non-*iid* distributions.

Another open problem that could be amenable to an LP-based attack is to determine the  $\Sigma_3^k$ -circuit complexity of inner product in case  $k = 3$ , as was originally asked by Golovnev, Kulikov, and Williams [2]. The best lower bound known is  $2^{n/3}$  [14], and one could hope to show an improved lower bound even relative to an *iid* distribution. Here the obvious challenge is that 3-CNFs are notoriously much more difficult (even NP-hard) to analyse than 2-CNFs. Our overall approach in this paper is still applicable even for  $k > 2$ . Namely, one needs to “merely” prove an analogue of our structure lemma (Lemma 7) for  $k$ -CNFs.

## 2 Upper bound

In this section, we prove the upper bound  $\alpha \leq 0.965\dots$  as claimed in Theorem 1. The circuit will be constructed in two parts. To explain this, we denote, for an input  $(x, y) \in \{0, 1\}^{2n}$  and a 2-bit pattern  $s \in \{0, 1\}^2$ , the fraction of occurrences of this pattern by

$$p_s(x, y) := \frac{1}{n} |\{i \in [n] : (x_i, y_i) = s\}|.$$

We use one  $\Sigma_3^2$ -circuit to accept every input  $(x, y) \in \text{IP}_n^{-1}(1)$  with  $p_{11}(x, y) \leq p$  where  $p$  is a carefully chosen threshold, and another  $\Sigma_3^2$ -circuit to accept those inputs with  $p_{11}(x, y) \geq p$ .

The following two lemmas (proved in Sections 2.1 and 2.2) record the two types of circuits we will construct. To state these lemmas, recall that a circuit  $C$  is *consistent* with  $\text{IP}_n$  if  $C(x, y) \leq \text{IP}_n(x, y)$  for all inputs  $(x, y)$ . We let  $\mathbb{H}(p) := -p \log p - (1-p) \log(1-p)$  denote the binary entropy function. Moreover, we let  $\mathbb{H}(X)$  denote the usual Shannon entropy of a random variable  $X$ . Finally, for  $p \in [0, 1]$ , we define a random variable  $X_p \in \{0, 1\}^2$  such that  $\Pr[X_p = 11] = p$  and  $\Pr[X_p = s] = (1-p)/3$  for  $s \in \{00, 01, 10\}$ .

► **Lemma 3.** *For every  $p \in [0, \frac{1}{2}]$  there exists a  $\Sigma_3^2$ -circuit of size  $2^{n\mathbb{H}(p)+o(n)}$  that is consistent with  $\text{IP}_n$  and that accepts all  $(x, y) \in \text{IP}_n^{-1}(1)$  with  $p_{11}(x, y) \leq p$ .*

► **Lemma 4.** *For every  $p \in [\frac{1}{4}, 1]$  there exists a  $\Sigma_3^2$ -circuit of size  $2^{\frac{1}{2}n\mathbb{H}(X_p)+o(n)}$  that is consistent with  $\text{IP}_n$  and that accepts all  $(x, y) \in \text{IP}_n^{-1}(1)$  with  $p_{11}(x, y) \geq p$ .*

The final  $\Sigma_3^2$ -circuit for  $\text{IP}_n$  is the OR of the two  $\Sigma_3^2$ -circuits above. It is easy to see that using any constant  $p \in (\frac{1}{4}, \frac{1}{2})$  we get a circuit of size  $2^{\beta n}$  with  $\beta < 1$ . We can further optimise the choice of  $p$  by equating the two circuit size expressions, solving for  $p$  numerically (using any numerical computation software), which comes to  $p := 0.3909\dots$ , and then plugging this value of  $p$  into the size expressions to get a circuit of size  $2^{0.965\dots n+o(n)}$ , as desired.

It remains to prove Lemmas 3 and 4, which we do in the rest of this section.

### 2.1 Proof of Lemma 3

In this lemma we focus on finding efficient  $\Sigma_3^2$ -circuits accepting inputs  $(x, y) \in \text{IP}^{-1}(1)$  with a small value of  $p_{11}(x, y) \leq p \leq 1/2$ . Given a subset  $I \subseteq [n]$ , define the *brute-force CNF* by

$$\phi_{\text{BF}}^{(I)} := \bigwedge_{i \in I} (x_i \wedge y_i) \wedge \bigwedge_{i \in [n] \setminus I} (\neg x_i \vee \neg y_i).$$

Note that  $\phi_{\text{BF}}^{(I)}$  accepts an input  $(x, y)$  iff  $I$  equals the set of all  $i$  such that  $(x_i, y_i) = (1, 1)$ . Hence, to accept every input with  $p_{11}(x, y) \leq p$ , our  $\Sigma_3^2$ -circuit will consider all suitable  $I$ :

$$C := \bigvee_{\substack{I \subseteq [n] \\ |I| \leq pn \\ |I| \text{ odd}}} \phi_{\text{BF}}^{(I)}. \tag{2}$$

The size of  $C$  is at most  $\binom{n}{\leq pn} \cdot O(n)$  where  $\binom{n}{\leq pn} := \sum_{i=0}^{pn} \binom{n}{i}$  can be estimated from above via Stirling’s approximation by  $2^{nH(p)+o(n)}$  for all  $p \leq 1/2$ . Finally, it is clear from the construction that  $C$  is consistent with  $\text{IP}_n$ . This concludes the proof of Lemma 3. ◀

### 2.2 Proof of Lemma 4

In this lemma we focus on finding efficient  $\Sigma_3^2$ -circuits accepting inputs  $(x, y) \in \text{IP}_n^{-1}(1)$  with a large value of  $p_{11}(x, y) \geq p \geq 1/4$ . To illustrate our idea, we first construct a circuit for a simpler related function, and then explain how to modify it to get circuits for  $\text{IP}_n$ .

**Simple warm-up circuit.** We first describe a circuit that computes the following partial function (which is consistent with  $\neg \text{IP}_n$ , but we will address this later):

$$f_n(x, y) := \begin{cases} 0 & \text{if } n \cdot p_{11}(x, y) \text{ is odd,} \\ 1 & \text{if } n \cdot p_s(x, y) \text{ is even for all } s \in \{0, 1\}^2, \text{ and } p_{11}(x, y) \geq p, \\ * & \text{otherwise.} \end{cases}$$

The interesting case here is when  $n$  is even, as otherwise  $f_n(x, y) \in \{0, *\}$  for all  $(x, y)$ . Let  $M \subseteq \binom{[n]}{2} := \{e \subseteq [n] : |e| = 2\}$  be a *perfect matching* of  $[n]$  (that is, partition of  $[n]$  into pairs). We define the *collision CNF* associated with  $M$  by

$$\phi_{\text{Coll}}^{(M)} := \bigwedge_{\{i, j\} \in M} (x_i \leftrightarrow x_j) \wedge (y_i \leftrightarrow y_j).$$

This is a 2-CNF since we can write an equivalence as  $a \leftrightarrow b \equiv (a \vee \neg b) \wedge (\neg a \vee b)$ . Note that a collision CNF accepts iff for every pair  $\{i, j\} \in M$  we have  $(x_i, y_i) = (x_j, y_j)$ . Hence it only accepts inputs where  $n \cdot p_s(x, y)$  is even for all  $s \in \{0, 1\}^2$ . Thus  $\phi_{\text{Coll}}^{(M)}$  is consistent with  $f_n$ .

To construct a  $\Sigma_3^2$ -circuit for  $f_n$ , it is enough, as discussed in Section 1.2, to design a feasible solution to the (LP) associated with  $f_n$ . (We note that the (LP) formulation works equally well for partial functions.) To this end, we calculate in the following claim (proved in Section 2.3) the probability that a *random* collision CNF accepts a fixed 1-input of  $f_n$ .

▷ **Claim 5.** Let  $(x, y) \in f_n^{-1}(1)$ . For a uniformly chosen perfect matching  $M \subseteq \binom{[n]}{2}$ ,

$$\Pr_M [\phi_{\text{Coll}}^{(M)}(x, y) = 1] \geq 2^{-\frac{1}{2}n\mathbb{H}(X_p)-o(n)} =: L(p).$$

## 51:6 Depth-3 Circuits for Inner Product

We now construct a feasible solution to (LP) for  $f_n$ . Let  $\Phi_{\text{Coll}}$  denote the set of all collision CNFs, one for each perfect matching of  $[n]$ . Consider the weight assignment corresponding to the uniform distribution over  $\Phi_{\text{Coll}}$ ; namely, set  $w_\phi := 1/|\Phi_{\text{Coll}}|$  for every  $\phi \in \Phi_{\text{Coll}}$  and  $w_\phi := 0$  for all the rest. Note that the objective function value is  $\sum_\phi w_\phi = 1$ . However, the assignment may not be feasible: for a covering constraint indexed by  $(x, y) \in f_n^{-1}(1)$ , we are only guaranteed a weak lower bound (much smaller than 1):

$$\sum_\phi w_\phi \phi(x, y) = \Pr_M [\phi_{\text{Coll}}^{(M)}(x, y) = 1] \geq L(p).$$

We can, however, transform this weight assignment into a feasible one by scaling all the weights up by a factor of  $1/L(p)$  (and truncating any resulting weight  $> 1$  to 1). In the scaled assignment, the objective function value is at most  $1/L(p)$ . We conclude (using Lemma 2) that  $f_n$  has a circuit of size  $O(n)/L(p) = 2^{\frac{1}{2}n\mathbb{H}(X_p)+o(n)}$ .

It remains to explain how a circuit of this size can also be constructed for  $\text{IP}_n$ .

**Actual circuit for  $\text{IP}$ .** To prove Lemma 4, we would like to use the  $\Sigma_3^2$ -circuit we constructed above for  $f_n$  to design a circuit for the partial function

$$\text{IP}_n^{(p)}(x, y) := \begin{cases} 0 & \text{if } n \cdot p_{11}(x, y) \text{ is even,} \\ 1 & \text{if } n \cdot p_{11}(x, y) \text{ is odd, and } p_{11}(x, y) \geq p, \\ * & \text{otherwise.} \end{cases}$$

Consider the following nondeterministic algorithm for  $\text{IP}_n^{(p)}$ . On input  $(x, y) \in \{0, 1\}^{2n}$ :

1. Nondeterministically guess a subset  $S \subseteq \{0, 1\}^2$  where  $11 \in S$ . The intention is that patterns in  $S$  should appear in  $(x, y)$  an odd number of times.
2. For each  $s \in S$ , guess a coordinate  $i(s) \in [n]$ .
3. For each  $s \in S$ , check that  $(x_{i(s)}, y_{i(s)}) = s$ . If not, reject.
4. Output the same as the function  $f_{n-|S|}$  on input  $(x_i, y_i)_{i \in [n] \setminus i(S)}$ .

It is straightforward to check that this computes  $\text{IP}_n^{(p)}$  correctly. (A minor technical detail is that when computing  $f_{n-|S|}$ , the  $p_{11}$  value may slightly drop because we remove one occurrence of the 11-pattern. However, this is not really a problem since the slight drop will not affect the asymptotics of the circuit size.) The question remains: How can it be implemented as a  $\Sigma_3^2$ -circuit? We do it as follows. Consider any guess outcome  $O := (S, (i(s))_{s \in S})$ . We can modify the circuit  $C$  for  $f_{n-|S|}$  (applied to the input bits  $(x_i, y_i)_{i \in [n] \setminus i(S)}$ ) to perform the check in Item 3 by adding to each 2-CNF in  $C$  the singleton terms  $(x_{i(s)} = s_1)$  and  $(y_{i(s)} = s_2)$  for all  $s = (s_1, s_2) \in S$ . Call the resulting circuit  $C_O$ . Our final  $\Sigma_3^2$ -circuit computes the OR of all circuits  $C_O$ . Since there are only  $O(n^4)$  many different guess outcomes, the resulting circuit is only a factor  $O(n^4)$  larger than our circuit for  $f_n$ . This concludes the proof of Lemma 4.  $\blacktriangleleft$

### 2.3 Proof of Claim 5

*Proof.* Write  $n!! := \prod_{i=0}^{\lfloor n/2 \rfloor} (n - 2i)$  for the double factorial. The number of perfect matchings on  $[n]$  is well-known to be given by  $(n - 1)!!$  when  $n$  is even. Therefore,  $(np_s - 1)!!$  gives the number of ways to match the coordinates with pattern  $s$ . We have

$$\Pr_M [\phi_{\text{Coll}}^{(M)}(x, y) = 1] = \frac{\prod_{s \in \{0,1\}^2} (np_s - 1)!!}{(n - 1)!!}. \quad (3)$$

Taking logarithms and using Stirling's approximation ( $\log n! = \frac{1}{2}n \log n - \frac{1}{2}n \pm o(n)$ ) we get

$$\begin{aligned} \log \Pr_M [\phi_{\text{Coll}}^{(M)}(x, y) = 1] &= \frac{1}{2} \sum_s np_s \log(np_s) - \frac{1}{2}n \log n \pm o(n) \\ &= \frac{1}{2}n \cdot \sum_s p_s \log p_s \pm o(n) \\ &= -\frac{1}{2}n \cdot \mathbb{H}(P) \pm o(n). \end{aligned}$$

Here  $P \in \{0, 1\}^2$  is the random variable defined by  $\Pr[P = s] = p_s$ . We ask: which random variable  $X \in \{0, 1\}^2$  maximises the entropy  $\mathbb{H}(X)$  subject to the constraint  $\Pr[X = 11] = p^*$ ? By the concavity of  $\mathbb{H}$  and symmetry (we can relabel outcomes without affecting the entropy), it is the random variable  $X_{p^*}$  such that

$$\Pr[X_{p^*} = 11] = p^*, \quad \Pr[X_{p^*} = 00] = \Pr[X_{p^*} = 10] = \Pr[X_{p^*} = 01] = (1 - p^*)/3.$$

The univariate map  $p^* \mapsto \mathbb{H}(X_{p^*})$  is also concave. It is maximised at  $p^* = 1/4$  (when  $X_{p^*}$  is uniform), and decreasing for  $p^* > 1/4$ . This means that, since  $1/4 \leq p \leq p_{11}$ , we have that  $\mathbb{H}(X_p) \geq \mathbb{H}(X_{p_{11}}) \geq \mathbb{H}(P)$ . Hence we obtain the claimed lower bound:

$$\log \Pr_M [\phi_{\text{Coll}}^{(M)}(x, y) = 1] \geq -\frac{1}{2}n \cdot \mathbb{H}(X_p) - o(n). \quad \triangleleft$$

### 3 Lower bound

In this section, we prove the lower bound  $\alpha \geq \log \frac{9}{5} = 0.847\dots$  as claimed in Theorem 1. We will follow the Dual LP strategy discussed in Section 1.2. Namely, we will choose a hard distribution over  $\text{IP}_n^{-1}(1)$  and then bound the acceptance probability of any 2-CNF consistent with  $\text{IP}_n$ . In fact, it is convenient to prove a slightly stronger statement and bound the acceptance probability of any 2-CNF consistent with  $\text{IP}_n$  or  $\neg\text{IP}_n$ . Indeed, we let  $\Phi_n$  denote the set of 2-CNFs consistent with  $\text{IP}_n$  or  $\neg\text{IP}_n$ .

**Hard distribution.** As the hard distribution, we consider the binomial distribution  $\mathcal{D}_p$  with parameter  $p \in (0, 1)$ , whose choice we will optimise later. That is,  $(X, Y) \sim \mathcal{D}_p$  is such that all bits are *iid*: they are independent and have identical distribution,  $\Pr[X_i = 1] = \Pr[Y_i = 1] = p$ . Note that  $\mathcal{D}_p$  is not in fact supported on  $\text{IP}_n^{-1}(1)$ , but it still places  $\Omega(1)$  probability mass on this set. Consequently, any  $\Sigma_3^2$ -circuit will have to cover  $\Omega(1)$  fraction of  $\mathcal{D}_p$  with its CNFs, so we can still use  $\mathcal{D}_p$  for proving a lower bound.

**Max-probability formulas.** Our goal will be to argue that any  $\phi \in \Phi_n$  has an acceptance probability dominated by one of two “maximum probability formulas” (*max-formulas*, for short). Namely, our first max-formula is the collision CNF (used in our upper bound in Section 2.2 and specialised here for one matching) and our second formula has a NAND constraint for each coordinate.

$$\text{1st max-formula: } \phi_{\text{Coll}}^{(n)} := \bigwedge_{i \in [n/2]} (x_{2i-1} \leftrightarrow x_{2i}) \wedge (y_{2i-1} \leftrightarrow y_{2i}) \quad \text{where } n \text{ is even,}$$

$$\text{2nd max-formula: } \phi_{\text{Nand}}^{(n)} := \bigwedge_{i \in [n]} (\neg x_i \vee \neg y_i).$$

Writing  $\Pr_{\mathcal{D}}[\phi] := \Pr_{(X,Y) \sim \mathcal{D}}[\phi(X, Y) = 1]$  for short, it is straightforward to see that

$$\Pr_{\mathcal{D}_p}[\phi_{\text{Coll}}^{(n)}] = (p^2 + (1-p)^2)^n \quad \text{and} \quad \Pr_{\mathcal{D}_p}[\phi_{\text{Nand}}^{(n)}] = (1-p^2)^n. \quad (4)$$

Equating these probabilities and solving for  $p$  yields our optimal choice  $p = p^* := 2/3$ . The following lemma states that these formulas have, for  $p = p^*$ , higher acceptance probabilities than any 2-CNF consistent with  $\text{IP}_n$  (or  $\neg\text{IP}_n$ ).

► **Lemma 6.**  $\Pr_{\mathcal{D}_{p^*}}[\phi] \leq M_{p^*}^{(n)} := \max \{ \Pr_{\mathcal{D}_{p^*}}[\phi_{\text{Coll}}^{(n)}], \Pr_{\mathcal{D}_{p^*}}[\phi_{\text{Nand}}^{(n)}] \}$  for any  $\phi \in \Phi_n$ .

Using Lemma 6 it is easy to complete our proof. We get for any  $\phi \in \Phi_n$ ,

$$\Pr_{\mathcal{D}_{p^*}}[\phi] \leq M_{2/3}^{(n)} = (1 - (2/3)^2)^n = 2^{-\log(9/5) \cdot n} = 2^{-0.847\dots n}.$$

As per Dual LP, the reciprocal of this probability yields the claimed circuit lower bound.

It remains to prove Lemma 6, which we do in the rest of this section.

### 3.1 Proof of Lemma 6

To help us analyse acceptance probabilities, we first prove a *structure lemma* for any consistent 2-CNF formula  $\phi$ . This lemma will find some “structured” formula  $\phi'$  that is (semantically) implied by  $\phi$ , denoted  $\phi \models \phi'$  (that is,  $\phi^{-1}(1) \subseteq \phi'^{-1}(1)$ ). The formula  $\phi'$  comes from a set of structured formulas  $\mathcal{S}_n$ , which we will carefully define in Section 3.2. For now, it suffices for us to know that each structured formula  $\phi^{(k)} \in \mathcal{S}_n$  only mentions variables among  $(x_i, y_i)_{i \in I}$  for some subset  $I \subseteq [n]$  of size  $|I| = k$  (possibly  $k \ll n$ ).

► **Lemma 7 (Structure lemma).** *Let  $\phi \in \Phi_n$  be a 2-CNF consistent with  $\text{IP}_n$  or  $\neg\text{IP}_n$ . Then there is some structured 2-CNF formula  $\phi^{(k)} \in \mathcal{S}_n$  such that  $\phi \models \phi^{(k)}$ .*

We can now formulate a “localised” version of Lemma 6 for structured formulas. It allows us to locally compare the acceptance probability of  $\phi^{(k)}$  with our max-formulas  $\phi_{\text{Coll}}^{(k)}$  and  $\phi_{\text{Nand}}^{(k)}$ , now defined naturally over  $k$  many coordinates. Our original definition of  $\phi_{\text{Coll}}^{(n)}$  was actually assuming  $n$  is even. For technical convenience, for odd  $n$ , we define  $\phi_{\text{Coll}}^{(n)} := \phi_{\text{Coll}}^{(n-1)} \wedge (x_n \leftrightarrow y_n)$ . The bounds in (4) continue to hold for this extended definition.

► **Lemma 8.**  $\Pr_{\mathcal{D}_{p^*}}[\phi^{(k)}] \leq M_{p^*}^{(k)} := \max \{ \Pr_{\mathcal{D}_{p^*}}[\phi_{\text{Coll}}^{(k)}], \Pr_{\mathcal{D}_{p^*}}[\phi_{\text{Nand}}^{(k)}] \}$  for any  $\phi^{(k)} \in \mathcal{S}_n$ .

Using Lemmas 7 and 8 (proved below) it is now easy to prove Lemma 6:

**Proof of Lemma 6.** We prove this by induction on  $n$ . The base case  $n = 0$  is vacuously true under the convention that  $\Pr[\phi_{\perp}] = M_{p^*}^{(0)} = 1$  for the empty formula  $\phi_{\perp}$ . For the inductive case  $n \geq 1$ , let  $\phi \in \Phi_n$  be arbitrary. Apply the structure lemma (Lemma 7) to find some  $\phi^{(k)} \in \mathcal{S}_n$  such that  $\phi \models \phi^{(k)}$ . Suppose for notational convenience  $\phi^{(k)}$  involves the first  $k \leq n$  coordinates. Let  $\mathcal{D}_{p^*}^{(k)}$  denote our binomial distribution over  $\{0, 1\}^{2k}$ . Then

$$\Pr_{\mathcal{D}_{p^*}^{(n)}}[\phi] \leq \sum_{\substack{a, b \in \{0, 1\}^k \\ \phi^{(k)}(a, b) = 1}} \Pr_{\mathcal{D}_{p^*}^{(k)}}[(a, b)] \cdot \Pr_{\mathcal{D}_{p^*}^{(n-k)}}[\phi|_{a, b}],$$

where  $\phi|_{a, b}$  is obtained from  $\phi$  by restricting the first  $k$  coordinates to values  $(a, b)$ . We note that restricting values in a formula consistent with  $\text{IP}_n$  might yield a formula consistent with  $\neg\text{IP}_{n-k}$  (and vice versa). We now apply Lemma 6 inductively for  $\phi|_{a, b}$  to conclude

$$\Pr_{\mathcal{D}_{p^*}^{(n)}}[\phi] \leq M_{p^*}^{(n-k)} \cdot \sum_{a, b} \Pr_{\mathcal{D}_{p^*}^{(k)}}[(a, b)] = M_{p^*}^{(n-k)} \cdot \Pr_{\mathcal{D}_{p^*}^{(k)}}[\phi^{(k)}] \leq M_{p^*}^{(n-k)} M_{p^*}^{(k)} = M_{p^*}^{(n)},$$

where the last inequality is Lemma 8 and the final equality follows from (4). ◀

The rest of this section is organised as follows. We first define our family of structured formulas  $\mathcal{S}_n$  in Section 3.2. Then we will prove Lemmas 7 and 8 in Sections 3.3 and 3.4.

### 3.2 Structured formulas in $\mathcal{S}_n$

We now proceed to define our family of structured formulas  $\mathcal{S}_n$ . The family will be closed under *symmetries of  $\text{IP}_n$* , as we now explain. The value of inner product  $\text{IP}_n$  remains unchanged if we permute its  $n$  coordinates (e.g., swap  $(x_i, y_i)$  with  $(x_j, y_j)$ ) or transpose two variables inside a single coordinate (i.e., swap  $(x_i, y_i)$  with  $(y_i, x_i)$ ). These permutations generate the group of symmetries of  $\text{IP}_n$ . We say that two CNFs  $\phi$  and  $\phi'$  are *isomorphic* if there is some symmetry  $\pi$  of  $\text{IP}_n$  that, when applied to  $\phi$  to yield  $\phi^\pi$ , makes the two formulas equivalent,  $\phi^\pi \equiv \phi'$ , that is, to accept the same set of inputs.

**Structured family  $\mathcal{S}_n$ .** To define  $\mathcal{S}_n$ , we list below its various members. Each formula is defined over some  $k \leq n$  pairs of literals  $L_k := \{\tilde{x}_1, \tilde{y}_1, \dots, \tilde{x}_k, \tilde{y}_k\}$  where  $\tilde{x}_i \in \{x_i, \neg x_i\}$  and  $\tilde{y}_i \in \{y_i, \neg y_i\}$ . Each item defines a type of 2-CNF with the understanding that each of its isomorphic copies is included in  $\mathcal{S}_n$ . See Figure 1 for illustrations. We start with two cases corresponding to our max-formulas.

1. **Nand** is  $\phi_{\text{Nand}}^{(1)} = (\neg x_1 \vee \neg y_1)$ . This is case  $n = 1$  of our second max-formula.
2. **Matching** is defined relative to a perfect matching  $M \subseteq \binom{L_k}{2}$  by

$$\phi_{\text{Match}}^{(k)} = \bigwedge_{\{\ell, \ell'\} \in M} (\ell \leftrightarrow \ell').$$

Note that this is a generalisation of our first max-formula (where the literals are positive and the perfect matching is more structured).

The final type of formula will be an extension of the following “ladder” formula

$$\psi^{(k)} = \bigwedge_{i=1}^{k-1} (\tilde{y}_i \leftrightarrow \tilde{x}_{i+1}) \quad \text{where } k \geq 2.$$

We also define two types of “terminal” constraints (where  $\ell, \ell' \in L_k$ ),

$$\begin{aligned} \text{Back-edge: } \psi_{\text{B}}^{\text{left}} &= (\tilde{x}_1 \leftrightarrow \ell), & \psi_{\text{B}}^{\text{right}} &= (\tilde{y}_k \leftrightarrow \ell') \quad \text{where } \ell \neq \tilde{x}_1 \text{ and } \ell' \neq \tilde{y}_k, \\ \text{Positive: } \psi_{\text{P}}^{\text{left}} &= (y_1 \rightarrow x_1), & \psi_{\text{P}}^{\text{right}} &= (x_k \rightarrow y_k). \end{aligned}$$

3. **Ladder** is given by choosing terminal types  $(L, R) \in \{\text{B}, \text{P}\}^2$  and defining

$$\phi_{LR}^{(k)} = \psi^{(k)} \wedge \psi_L^{\text{left}} \wedge \psi_R^{\text{right}}.$$

► **Remark 9.** It can be shown that this list is *irredundant* in that, for each type, there is a formula  $\phi^{(k)} \in \mathcal{S}_n$  of that type and  $\phi \in \Phi_n$  such that  $\phi \models \phi^{(k)}$  but  $\phi \not\models \phi'$  for every  $\phi' \in \mathcal{S}_n$  of type different than  $\phi^{(k)}$ . This means that we need all three types for our structure lemma.

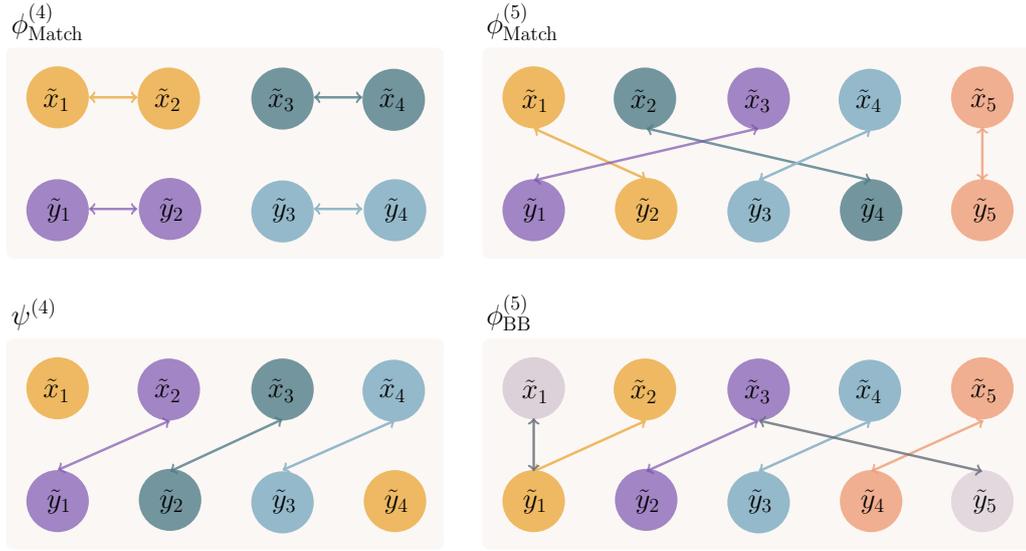
### 3.3 Proof of Structure lemma (Lemma 7)

In the proof of Lemma 7, we use the standard notion of an implication graph of a 2-CNF.

**Implication graphs.** Given a 2-CNF  $\phi$  over  $k$  variables  $\{x_1, y_1, \dots, x_k, y_k\}$ , its *implication graph*  $G_\phi = (V, E)$  is the directed graph given by

$$\begin{aligned} V &:= \{x_1, \neg x_1, y_1, \neg y_1, \dots, x_k, \neg x_k, y_k, \neg y_k\}, \\ E &:= \{(u, v) \in V^2 : u \neq v \text{ and } \phi \models (u \rightarrow v)\}. \end{aligned}$$

## 51:10 Depth-3 Circuits for Inner Product



■ **Figure 1** Examples of Matching and Ladder CNFs.

We note that implication graphs are sometimes defined more syntactically: For each clause  $(u \vee v)$  of  $\phi$ , include the edges  $(\neg u, v)$  and  $(\neg v, u)$  in  $G_\phi$ , and moreover, for each singleton clause  $(u)$  of  $\phi$ , include the edges  $(v, u)$  in  $G_\phi$  for all  $v$ . Taking the transitive closure (add edge  $(u, v)$  if there is a directed path from  $u$  to  $v$ ) of this graph yields the graph in our (semantic) definition above.

We call a strongly connected component of  $G_\phi$  a *strong-component* for short. We say that a variable  $x_i$  is fixed by  $\phi$  if there is some  $b \in \{0, 1\}$  such that for every  $(x, y) \in \phi^{-1}(1)$  we have  $x_i = b$ . The following lemma will be used several times.

► **Lemma 10.** *Let  $\phi \in \Phi_n$  and suppose  $y_1$  lies in a strong-component of size 1 in  $G_\phi$ . Then we have  $\phi \models x_1 \rightarrow \tilde{y}_1$  for some  $\tilde{y}_1 \in \{y_1, \neg y_1\}$ .*

**Proof.** We may assume that  $y_1$  is not fixed by  $\phi$ , as otherwise the lemma is trivially true. We assume that  $\phi \not\models x_1 \rightarrow \neg y_1$  and hope to show  $\phi \models x_1 \rightarrow y_1$ . Thus, there is some satisfying assignment  $(x', y') \in \phi^{-1}(1)$  such that  $(x'_1, y'_1) = (1, 1)$ . Denote by  $N_{\text{in}} \subseteq V$  the in-neighbours of  $y_1$ , that is, all the literals from which there exists an edge (equivalently, directed path, as  $G_\phi$  is transitively closed) to  $y_1$ . Note that  $\{\ell, \neg \ell\} \not\subseteq N_{\text{in}}$  for every literal  $\ell$ , as otherwise one of  $\ell$  or  $\neg \ell$  would always be set to 1, forcing  $y_1$  to always be 1, contradicting that  $y_1$  is not fixed. Modify  $(x', y')$  by setting literals in  $N_{\text{in}}$  to 0. By the properties listed above, it follows that the new assignment, call it  $(x'', y'')$ , still satisfies  $\phi$ . Moreover,  $(x'', y'')$  has the property that we may flip the value of all the literals in the strong-component of  $y_1$  – which is just  $y_1$  itself – and still remain a satisfying assignment. Since we can flip  $y_1$  in isolation, we must have that  $x''_1 = 0$  (otherwise we would change the parity of the 11 pattern, which contradicts  $\phi \in \Phi_n$ ). But since  $x'_1 = 1$  we must have that  $x_1 \in N_{\text{in}}$ , meaning that  $(x_1, y_1)$  is an edge, and hence  $\phi \models x_1 \rightarrow y_1$ , as desired. ◀

We now proceed to prove Lemma 7 in two cases by considering  $G_\phi$  for  $\phi \in \Phi_n$ .

**Case 1.** *Every strong-component is of size 1.* Applying Lemma 10 twice, the second time with roles of  $x_1$  and  $y_1$  swapped, we learn that  $\phi \models x_1 \rightarrow \tilde{y}_1$  and  $\phi \models y_1 \rightarrow \tilde{x}_1$ . If  $\phi \models x_1 \rightarrow \neg y_1$  or  $\phi \models y_1 \rightarrow \neg x_1$  holds then we have  $\phi \models \phi_{\text{Nand}}^{(1)}$ , as desired. In the remaining case, both  $\phi \models x_1 \rightarrow y_1$  and  $\phi \models y_1 \rightarrow x_1$  hold, which implies  $\phi \models \phi_{\text{Match}}^{(1)}$ .

**Case 2.** *There exists a strong-component of size at least 2.* Suppose by symmetry that  $y_1$  lies in a strong-component of size at least 2. If  $y_1$  is bidirectionally connected to  $\tilde{x}_1$ , that is,  $\phi \models (y_1 \leftrightarrow \tilde{x}_1)$ , then this means that  $\phi \models \phi_{\text{Match}}^{(1)}$  and we are done.

Assume henceforth that  $y_1$  is bidirectionally connected to some literal other than  $\tilde{x}_1$ , say by symmetry  $y_1 \leftrightarrow \tilde{x}_2$ . Consider  $y_2$ : is it bidirectionally connected to a literal in coordinate greater than 2? If yes, say by symmetry  $y_2 \leftrightarrow \tilde{x}_3$ . Consider  $y_3$ , etc. By this “unravelling” process, we are exposing the bidirectional edges of a ladder formula  $\psi^{(k)}$ . This process must eventually end at step  $k \leq n$  in one of the following two cases.

- **Subcase 2-1:**  $y_k$  is bidirectionally connected to some literal  $\ell'$  in coordinate  $\leq k$ . Here we have  $\phi \models (y_k \leftrightarrow \ell') = \psi_{\text{B}}^{\text{right}}$ .
- **Subcase 2-2:**  $y_k$  lies in a singleton strong-component. In this case, we apply Lemma 10 to learn that  $\phi \models x_k \rightarrow \tilde{y}_k$ . If  $\phi \models x_k \rightarrow \neg y_k$ , then we would have found a copy of  $\phi_{\text{Nand}}^{(1)}$  in coordinate  $k$  and we are done. Otherwise  $\phi \models x_k \rightarrow y_k$ , which means  $\phi \models \psi_{\text{P}}^{\text{right}}$ .

That is, in both cases (if we did not outright prove the lemma) we found either  $\phi \models \psi_{\text{B}}^{\text{right}}$  or  $\phi \models \psi_{\text{P}}^{\text{right}}$ . By a similar argument, we can start unravelling edges starting at  $x_1$  to find either  $\phi \models \psi_{\text{B}}^{\text{left}}$  or  $\phi \models \psi_{\text{P}}^{\text{left}}$ . This will allow us to terminate the left side of the ladder, which completes the proof that  $\phi \models \phi_{\text{LR}}^{(k)}$ .

### 3.4 Proof of Lemma 8

We show the inequalities for every  $\phi \in \mathcal{S}_n$ .

- $\phi_{\text{Nand}}^{(1)}$ : This is true by definition of  $M_p^{(1)}$ .
- $\phi_{\text{Match}}^{(k)}$ : First note that the structure of the perfect matching for  $\phi_{\text{Match}}^{(k)}$  will not change the acceptance probability because all input bits are *iid*. Moreover, when both  $\ell$  and  $\ell'$  are positive,  $\Pr[\ell \leftrightarrow \ell'] = p^2 + (1-p)^2$ ; otherwise,  $\Pr[\ell \leftrightarrow \ell'] = \max\{2p(1-p), p^2 + (1-p)^2\} \leq p^2 + (1-p)^2$  for all  $p \in [0, 1]$ . Therefore, we have that  $\Pr_{\mathcal{D}_p}[\phi_{\text{Match}}^{(k)}] \leq \Pr_{\mathcal{D}_p}[\phi_{\text{Coll}}^{(k)}]$ .
- $\phi_{\text{BB}}^{(k)}$ : We show in the above that  $\Pr[\ell \leftrightarrow \ell'] \leq p^2 + (1-p)^2$  for any literals  $\ell$  and  $\ell'$ ; we can similarly show that, for any literals  $\ell, \ell'$  and  $\ell''$ ,  $\Pr[\ell \leftrightarrow \ell', \ell \leftrightarrow \ell''] \leq p^3 + (1-p)^3$ . Replacing all literals in  $\phi_{\text{BB}}^{(k)}$  by their positive analogues to get a new CNF  $\phi$ , we have that  $\Pr_{\mathcal{D}_p}[\phi_{\text{BB}}^{(k)}] \leq \Pr_{\mathcal{D}_p}[\phi]$ . Let  $M$  be the perfect matching associated with  $\phi$ . Define  $M' := M \cup \{(x_1, y_k)\}$ . Observe that  $M'$  is a perfect matching for all  $2k$  literals. Let  $\phi'$  be the matching CNF constructed from  $M'$ . Let  $P$  be the acceptance probability of  $\phi$ . We know that  $\Pr_{\mathcal{D}_p}[\phi'] = P \cdot \frac{[(1-p)^2 + p^2]^3}{[(1-p)^3 + p^3]^2} \geq P$  since  $\frac{[(1-p)^2 + p^2]^3}{[(1-p)^3 + p^3]^2} \geq 1$  for  $p \in [0, 1]$ .
- $\phi_{\text{PP}}^{(k)}$ : Similarly, we can replace all literals in  $\phi_{\text{PP}}^{(k)}$  with their positive analogues and get  $\phi$ . Let  $M$  be the perfect matching associated with  $\phi$ . Define  $M' := M \cup \{(x_1, y_k)\}$ . Observe that  $M'$  is a perfect matching for all  $2k$  literals. Let  $\phi'$  be the matching CNF constructed from  $M'$ . Let  $P$  be the acceptance probability of  $\phi$ . If  $k = 2$  then we have that  $P = (1-p)^2 + p^4 = [(1-p)^2 + p^2]^2 = \Pr_{\mathcal{D}_p}[\phi']$  for  $p = \frac{2}{3}$ . If  $k > 2$ , we know that  $\Pr_{\mathcal{D}_p}[\phi'] = P \cdot \frac{[(1-p)^2 + p^2]^3}{[(1-p)^2 + p^3]^2} > P$  since  $\frac{[(1-p)^2 + p^2]^3}{[(1-p)^2 + p^3]^2} > 1$  for  $p = \frac{2}{3}$ .
- $\phi_{\text{BP}}^{(k)}$ : As we have seen before, we can replace all literals in  $\phi_{\text{BP}}^{(k)}$  with their positive analogues and get  $\phi$ . Let  $M$  be the perfect matching associated with  $\phi$ . Define  $M' := M \cup \{(x_1, y_k)\}$ . Observe that  $M'$  is a perfect matching for all  $2k$  literals. Let  $\phi'$  be the

matching CNF constructed from  $M'$ . Let  $P$  be the acceptance probability of  $\phi$ . If  $k = 2$  then we have that  $P = (1 - p)^3 + p^4 < [(1 - p)^2 + p^2]^2 = \Pr_{\mathcal{D}_p}[\phi']$  for  $p = \frac{2}{3}$ . If  $k > 2$ , we know that  $\Pr_{\mathcal{D}_p}[\phi'] = P \cdot \frac{((1-p)^2+p^2)^3}{((1-p)^2+p^3)[(1-p)^3+p^3]} > P$  since  $\frac{((1-p)^2+p^2)^3}{((1-p)^2+p^3)[(1-p)^3+p^3]} > 1$  for  $p = \frac{2}{3}$ .

---

## References

- 1 Peter Frankl, Svyatoslav Gryaznov, and Navid Talebanfard. A Variant of the VC-Dimension with Applications to Depth-3 Circuits. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215, pages 72:1–72:19, Dagstuhl, 2022. Schloss Dagstuhl. doi:10.4230/LIPIcs.ITCS.2022.72.
- 2 Alexander Golovnev, Alexander S. Kulikov, and R. Ryan Williams. Circuit Depth Reductions. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185, pages 24:1–24:20, Dagstuhl, 2021. Schloss Dagstuhl. doi:10.4230/LIPIcs.ITCS.2021.24.
- 3 J. Håstad, S. Jukna, and P. Pudlák. Top-down lower bounds for depth-three circuits. *Computational Complexity*, 5(2):99–112, June 1995. doi:10.1007/bf01268140.
- 4 Suichi Hirahara. A duality between depth-three formulas and approximation by depth-two. Technical report, arXiv, 2017. arXiv:1705.03588.
- 5 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, December 2001. doi:10.1006/jcss.2001.1774.
- 6 Rahul Jain and Hartmut Klauck. The partition bound for classical communication complexity and query complexity. In *Proceedings of the 25th Conference on Computational Complexity (CCC)*, pages 247–258. IEEE, 2010. doi:10.1109/CCC.2010.31.
- 7 Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012.
- 8 Mauricio Karchmer, Eyal Kushilevitz, and Noam Nisan. Fractional covers and communication complexity. *SIAM Journal on Discrete Mathematics*, 8(1):76–92, February 1995. doi:10.1137/s0895480192238482.
- 9 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- 10 Troy Lee and Adi Shraibman. *Lower Bounds in Communication Complexity*, volume 3. Now Publishers, 2007. doi:10.1561/04000000040.
- 11 L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975. doi:10.1016/0012-365X(75)90058-8.
- 12 R. Paturi, M. E. Saks, and F. Zane. Exponential lower bounds for depth three boolean circuits. *computational complexity*, 9(1):1–15, 2000. doi:10.1007/PL00001598.
- 13 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for  $k$ -SAT. *Journal of the ACM*, 52(3):337–364, May 2005. doi:10.1145/1066100.1066101.
- 14 Ramamohan Paturi, Pavel Pudlak, and Francis Zane. Satisfiability coding lemma. *Chicago Journal of Theoretical Computer Science*, 5(1):1–19, 1999. doi:10.4086/cjtcs.1999.011.
- 15 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977*, pages 162–176, Berlin, Heidelberg, 1977. Springer Berlin Heidelberg.
- 16 Emanuele Viola. On the power of small-depth computation. *Foundations and Trends in Theoretical Computer Science*, 5(1):1–72, 2009. doi:10.1561/04000000033.

# On Polynomial-Time Decidability of $k$ -Negations Fragments of FO Theories (Extended Abstract)

Christoph Haase  

University of Oxford, UK

Alessio Mansutti  

IMDEA Software Institute, Madrid, Spain

Amaury Pouly 

University of Oxford, UK

Université Paris Cité, CNRS, IRIF, France

---

## Abstract

---

This paper introduces a generic framework that provides sufficient conditions for guaranteeing polynomial-time decidability of fixed-negation fragments of first-order theories that adhere to certain fixed-parameter tractability requirements. It enables deciding sentences of such theories with arbitrary existential quantification, conjunction and a fixed number of negation symbols in polynomial time. It was recently shown by Nguyen and Pak [*SIAM J. Comput.* 51(2): 1–31 (2022)] that an even more restricted such fragment of Presburger arithmetic (the first-order theory of the integers with addition and order) is NP-hard. In contrast, by application of our framework, we show that the fixed negation fragment of weak Presburger arithmetic, which drops the order relation from Presburger arithmetic in favour of equality, is decidable in polynomial time.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** first-order theories, arithmetic theories, fixed-parameter tractability

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.52

**Funding** This work is part of a project that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARIAT).



## 1 Introduction

It is well-known that even the simplest first-order theories are computationally difficult to decide [12]. In particular, it follows from a result of Stockmeyer that every theory with a non-trivial predicate such as equality is PSPACE-hard to decide [24]. Even when restricting to existential fragments or fragments with a fixed number of quantifier alternations, deciding such fragments is NP-hard at best. There are two further kinds of restrictions that may lead to tractability. First, restricting the Boolean structure of the matrix of formulae in prenex form yields tractable fragments of, e.g., the Boolean satisfiability problem. For instance, the Horn and XOR-fragments of propositional logic are decidable in polynomial time, and this even applies to quantified Boolean Horn formulae, see e.g. [7]. Second, restricting the number of variables can also lead to tractable fragments of a first-order theory, especially for structures over infinite domains such as Presburger arithmetic, the first-order theory of the structure  $(\mathbb{Z}, 0, 1, +, \leq)$ . While the existential fragment of Presburger arithmetic is NP-complete in general [5, 25], it becomes polynomial-time decidable when additionally fixing the number of variables [22]; this is a consequence of polynomial-time decidability of integer programming when the dimension is fixed [18]. Already when moving to an  $\exists\forall$  quantifier prefix, Presburger arithmetic becomes NP-hard [23]. On the first sight, this result seems to preclude any possibility of further restrictions that may lead to tractable fragments of



© Christoph Haase, Alessio Mansutti, and Amaury Pouly;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 52; pp. 52:1–52:14

Leibniz International Proceedings in Informatics



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

Presburger arithmetic. However, another tractable fragment was identified in the context of investigating the complexity of the classical *Frobenius problem*. Given positive integers  $a_1, \dots, a_n \in \mathbb{N}$ , this problem is to determine the largest integer that cannot be obtained as a non-negative linear combination of the  $a_i$ , which is called the *Frobenius number*. For any fixed  $n > 0$ , deciding whether the Frobenius number exceeds a given threshold can be reduced to the so-called *short fragment* of Presburger arithmetic, a highly restricted fragment in which everything, the number of atomic formulae and the number of variables (and *a fortiori* the number of quantifier alternations), is fixed – except for the coefficients of variables appearing in linear terms of atomic inequalities. Kannan [15] showed that the  $\forall^k \exists^\ell$ -fragment of short Presburger arithmetic is decidable in polynomial time for all fixed  $k, \ell$ , which implies that the decision version of the Frobenius problem is in polynomial time for fixed  $n$ . However, in a recent breakthrough, Nguyen and Pak showed that there are fixed  $k, \ell, m$  such that the  $\exists^k \forall^\ell \exists^m$ -fragment of short Presburger arithmetic is NP-hard, and adding further (fixed) quantifier alternations allows the logic to climb the polynomial hierarchy [20].

The main contribution of this paper is to develop an algorithmic framework that enables us to show that *fixed negation fragments* of certain first-order theories are decidable in polynomial time. Formulae in this fragment are generated by the following grammar, where  $\Psi$  are atomic formulae of the underlying first-order theory, and an arbitrary but a priori fixed number of negation symbols is allowed to occur:

$$\Phi ::= \exists x \Phi \mid \neg \Phi \mid \Phi \wedge \Phi \mid \Psi.$$

We give sufficient conditions for the fixed negation fragment of a first-order theory to be decidable in polynomial time. Observe that this fragment is more permissive than the “short fragments” of Kannan, as it allows for an unbounded number of quantified variables and an unbounded number of conjunctions. However, it implicitly fixes the number of quantifier alternations as well as the number of disjunctions.

As a main application of our framework, we show that, unlike full Presburger arithmetic, the fixed negation fragment of weak Presburger arithmetic (weak PA) is polynomial-time decidable. Weak PA is the strictly less expressive substructure  $(\mathbb{Z}, 0, 1, +, =)$  (which is, in fact, the structure Presburger studied in his seminal article [21]). It was recently shown that weak PA has the same complexity as Presburger arithmetic [9]. Bodirsky et al. showed that the weak PA fragment of (an unbounded number of) existential linear Horn equations  $\bigwedge_{i \in I} (A_i \cdot \mathbf{x} = b_i) \rightarrow (C_i \cdot \mathbf{x} = d_i)$  over  $\mathbb{Z}$  can be decided in PTIME [4]. It follows from the generic results in this paper that the quantified versions of those formulae with the number of quantifier alternations and  $|I|$  fixed is also polynomial-time decidable. This is the best possible such result, since one can show that, for  $I$  unbounded, the  $\exists \forall$  fragment of linear Horn equations in 2 variables is NP-hard (a proof of this is given in Appendix A). Our framework not only allows for deciding satisfiability (and validity) of fixed negation formulae of Weak PA in PTIME, but also to compute a representation of the set of solutions of a given formula.

Our algorithmic framework is parametric on a concrete representation of the sets definable within the first-order theory  $\mathcal{T}$  under consideration and only requires a sensible representation of solution sets for conjunctions of atomic formulae. From this representation, the framework guides us to the definition of a companion structure  $\mathcal{R}$  for the theory  $\mathcal{T}$  in which function symbols and relations in  $\mathcal{R}$  are interpreted as reductions from parametrised complexity theory, e.g. UXP reductions, see e.g. [11, Chapter 15]. By requiring mild conditions on the types of reductions and parameters that the functions and relations in  $\mathcal{R}$  must have, we are able to give a general theorem for the tractability of the fixed negation satisfiability

and entailment problems for  $\mathcal{T}$ . One technical issue we show how to overcome in a general way is how to treat negation, which is especially challenging when the initial representation provided to the framework is not closed under complementation (as it is the case in our treatment of weak PA). We resolve this issue by reducing it to the problem of completing a prelattice under relative complement, which we analyse computationally. Our main source of inspiration here is the notion of the so-called *difference normal form* of propositional logic, a rather unorthodox normal form introduced by Hausdorff [13, Ch. 1§5].

## 2 Preliminaries

We assume familiarity with basic concepts from first-order logic, set theory and abstract algebra. This section focuses on notation and simple definitions that might be non-standard to some readers.

**Sets and functions.** We write  $\text{seq}(A)$  for the set of all finite tuples over a set  $A$ , and denote by  $()$  the empty tuple. This definition corresponds to the standard notion of Kleene star  $A^*$  of a set  $A$ . The discrepancy in notation is introduced to avoid writing  $(\Sigma^*)^*$  for the domain of all tuples of finite words over an alphabet  $\Sigma$ , as in formal languages the Kleene star comes equipped with the axiom  $(\Sigma^*)^* = \Sigma^*$ . We denote this domain by  $\text{seq}(\Sigma^*)$ .

We write  $f : \subseteq X \rightarrow Y$  (resp.  $f : X \rightarrow Y$ ) to denote a *partial* (resp. *total*) function from  $X$  to  $Y$ . The domain of  $f$  is denoted with  $\text{dom}(f)$ . We write  $\text{id}_A : A \rightarrow A$  for the identity function on  $A$ . Given  $f : \subseteq A \rightarrow B$ ,  $g : \subseteq B \rightarrow C$  and  $h : \subseteq D \rightarrow E$ , we denote by  $(g \circ f) : \subseteq A \rightarrow C$  and  $(f \times h) : \subseteq A \times D \rightarrow B \times E$  the *composition* and the *Cartesian product* of functions.

**Structures with indexed families of functions.** We consider an expansion to the traditional definition of structure from universal algebra that accommodate for a potentially infinite number of functions. As usual, a *structure*  $\mathcal{A} = (A, \sigma, I)$  consists of a *domain*  $A$  (a set), a *signature*  $\sigma$ , and an *interpretation function*  $I$ ; however in our case the signature is a quadruple  $\sigma = (\mathcal{F}, \mathcal{G}, \mathcal{R}, \text{ar})$  containing not only a set of *function symbols*  $\mathcal{F}$ , a set of *relation symbols*  $\mathcal{R}$ , and the *arity function*  $\text{ar} : \mathcal{F} \uplus \mathcal{R} \uplus \mathcal{G} \rightarrow \mathbb{N}$ , but also a set of (indexed) *families of function symbols*  $\mathcal{G}$ . Each element of  $\mathcal{G}$  is a pair  $(g, X)$  where  $g$  is a function symbol and  $X$  is a countable set of indices. The interpretation function  $I$  associates to every  $f \in \mathcal{F}$  a map  $f^{\mathcal{A}} : A^{\text{ar}(f)} \rightarrow A$ , to every  $(g, X) \in \mathcal{G}$  a map  $g^{\mathcal{A}} : X \times A^{\text{ar}(g)} \rightarrow A$ , and to every  $R \in \mathcal{R}$  a relation  $R^{\mathcal{A}} \subseteq A^{\text{ar}(R)}$  which we often see as a function  $R^{\mathcal{A}} : A^{\text{ar}(R)} \rightarrow \{\top, \perp\}$ .

The standard notions of *homomorphism*, *embedding* and *isomorphism of structures*, as well as the notions of *congruence for a structure* and *quotient structure* extend in a natural way to structures having families of functions. For instance, a *homomorphism* from  $\mathcal{A} = (A, \sigma, I)$  into  $\mathcal{B} = (B, \sigma, J)$  is a map  $h : A \rightarrow B$  that *preserves* all functions, families of functions and relations; so in particular given  $(g, X) \in \mathcal{G}$ , the map  $h$  satisfies  $g^{\mathcal{B}}(x, h(a_1), \dots, h(a_{\text{ar}(g)})) = h(g^{\mathcal{A}}(x, a_1, \dots, a_{\text{ar}(g)}))$  for every  $x \in X$  and  $a_1, \dots, a_{\text{ar}(g)} \in A$ .

We denote structures in calligraphic letters  $\mathcal{A}, \mathcal{B}, \dots$  and their domains in capital letters  $A, B, \dots$ . When the arity function  $\text{ar}$  and the interpretation  $I$  are clear from the context, we write  $(A, f_1^{\mathcal{A}}, \dots, f_j^{\mathcal{A}}, (g_1^{\mathcal{A}}, X_1), \dots, (g_\ell^{\mathcal{A}}, X_\ell), R_1^{\mathcal{A}}, \dots, R_k^{\mathcal{A}})$ , and often drop the superscript  $\mathcal{A}$ , for a structure  $\mathcal{A} = (A, \sigma, I)$  with  $\sigma = (\{f_1, \dots, f_j\}, \{(g_1, X_1), \dots, (g_\ell, X_\ell)\}, \{R_1, \dots, R_k\}, \text{ar})$ . A structure  $\mathcal{A} = (A, \sigma, I)$  is said to be an *algebra of sets* whenever  $A$  is a family of sets and the symbols in  $\sigma$  are taken from  $\{\emptyset, \cup, \cap, \setminus, \subseteq, =\}$  and interpreted as the canonical operations on sets. Since  $\mathcal{A}$  is a structure, the set  $A$  is closed under all set operations in  $\sigma$ .

► **Example 1.** To understand the notion of families of function, consider the structure  $\mathcal{A} = (\mathbb{Z}, (\text{mul}, \mathbb{N}))$ . Here, the family of functions  $(\text{mul}, \mathbb{N})$ , interpreted as  $\text{mul}^{\mathcal{A}}(n, x) = n \cdot x$  for all  $n \in \mathbb{N}$  and  $x \in \mathbb{Z}$ , uniformly define multiplication by a non-negative constant  $n$ .

**First-order theories (finite tuples semantics).** The first-order (FO) language of the signature  $\sigma = (\mathcal{F}, \mathcal{G}, \mathcal{R}, ar)$  is the set of formulae  $\Phi, \Psi, \dots$  built from the grammar

$$\Phi, \Psi := r(t_1, \dots, t_{ar(r)}) \mid \neg\Phi \mid \Phi \wedge \Psi \mid \exists x. \Phi, \quad t := x \mid f(t_1, \dots, t_{ar(f)}) \mid g(i, t_1, \dots, t_{ar(g)}),$$

where  $x \in \mathbb{V}$  is a first-order variable,  $r \in \mathcal{R}$ ,  $f \in \mathcal{F}$ ,  $(g, X) \in \mathcal{G}$  and  $i \in X$  (more precisely,  $i$  belongs to a representation of  $X$ , see Section 3). Lexemes of the form  $r(t_1, \dots, t_{ar(r)})$  are the *atomic formulae* of the language. Throughout the paper, we implicitly assume an order on the variables in  $\mathbb{V}$ , and write  $x_j$  for the  $j$ -th variable (indexed from 1).

For our purposes it comes handy to define the FO theory of a structure  $\mathcal{A} = (A, \sigma, I)$  using tuples instead of the more standard approach of having maps from variables to values. The two definitions are equivalent. Given an atomic formula  $r(t_1, \dots, t_{ar(r)})$  having  $x_n$  as the largest appearing variable, we write  $\llbracket r(t_1, \dots, t_{ar(r)}) \rrbracket_{\mathcal{A}} \subseteq A^n$  for the set of  $n$ -tuples, corresponding to values of the first  $n$  variables, that makes  $r(t_1, \dots, t_{ar(r)})$  true under the given interpretation  $I$ . Let  $\mathbf{I} := \{(i_1, \dots, i_k) \in \text{seq}(\mathbb{N}) : i_1, \dots, i_k \text{ all distinct}\}$ . The *first-order theory* of  $\mathcal{A}$  is the structure  $\text{FO}(\mathcal{A}) := (\llbracket \mathcal{A} \rrbracket_{\text{FO}}, \perp, \top, \vee, \wedge, -, (\pi, \mathbf{I}), (\pi^{\forall}, \mathbf{I}), \leq)$ , where:

1.  $\llbracket \mathcal{A} \rrbracket_{\text{FO}}$  is the least set that contains  $\llbracket r(t_1, \dots, t_{ar(r)}) \rrbracket_{\mathcal{A}}$ , for each formula  $r(t_1, \dots, t_{ar(r)})$ , and that is closed under the functions  $\perp, \top, \vee, \wedge, -, (\pi, \mathbf{I})$  and  $(\pi^{\forall}, \mathbf{I})$ , defined below;
2.  $\perp$  is interpreted as  $\emptyset$ ,  $\top$  is interpreted as  $\{\}$ , and given  $S \subseteq A^n$  and  $T \subseteq A^m$ ,

$$\begin{aligned} S \vee T &:= \{(a_1, \dots, a_{\max(n,m)}) : (a_1, \dots, a_n) \in S \text{ or } (a_1, \dots, a_m) \in T\}, \\ S \wedge T &:= \{(a_1, \dots, a_{\max(n,m)}) : (a_1, \dots, a_n) \in S \text{ and } (a_1, \dots, a_m) \in T\}, \\ S - T &:= \{(a_1, \dots, a_{\max(n,m)}) : (a_1, \dots, a_n) \in S \text{ and } (a_1, \dots, a_m) \notin T\}, \\ \pi((i_1, \dots, i_k), X) &:= \{\gamma \in A^n : \text{there is } \mathbf{a} \in A^k \text{ such that } \gamma[(i_1, \dots, i_k) \leftarrow \mathbf{a}] \in X\}, \\ \pi^{\forall}((i_1, \dots, i_k), X) &:= \{\gamma \in A^n : \text{for every } \mathbf{a} \in A^k, \gamma[(i_1, \dots, i_k) \leftarrow \mathbf{a}] \in X\}, \\ S \leq T &\text{ if and only if } S \times A^m \subseteq T \times A^n, \end{aligned}$$

where  $\gamma[(i_1, \dots, i_k) \leftarrow \mathbf{a}]$  is the tuple obtained from the  $n$ -tuple  $\gamma$  by replacing its  $i_j$ -th component with the  $j$ -th component of  $\mathbf{a}$ , for every  $j \in [1, \min(k, n)]$ .

The semantics  $\llbracket \cdot \rrbracket_{\mathcal{A}}$  of the first-order language of  $\sigma$  is extended to non-atomic formulae via  $\text{FO}(\mathcal{A})$ . As usual, for negation, conjunction and existential quantification, we have  $\llbracket \neg\Phi \rrbracket_{\mathcal{A}} := \top - \llbracket \Phi \rrbracket_{\mathcal{A}}$ ,  $\llbracket \Phi \wedge \Psi \rrbracket_{\mathcal{A}} := \llbracket \Phi \rrbracket_{\mathcal{A}} \wedge \llbracket \Psi \rrbracket_{\mathcal{A}}$ , and  $\llbracket \exists x_i. \Phi \rrbracket_{\mathcal{A}} := \pi((i), \llbracket \Phi \rrbracket_{\mathcal{A}})$ . We remark that  $\text{FO}(\mathcal{A})$  contains operators whose syntactic counterpart is absent from the FO language of  $\sigma$ , such as the universal projection  $\pi^{\forall}$ . This is done for algorithmic purposes, as the framework we introduce in Section 4 treats these operators as first-class citizens.

We let  $\text{AC}(\sigma)$  be the set of all conjunctions of atomic formulae in the FO language of  $\sigma$ .

**Fixed negation fragments.** Fix  $k \in \mathbb{N}$ . The  $k$ -negations fragment of the FO language of a signature  $\sigma$  is the set of all formulae having at most  $k$  negations  $\neg$ . Note that, following the grammar of FO languages provided above, this restriction also bounds the number of disjunctions and alternations between existential and universal quantifiers that formulae can have. Given a structure  $\mathcal{A} = (A, \sigma, I)$ , we are interested in the following problem:

$k$  NEGATIONS SATISFIABILITY: given a formula  $\Phi$  with at most  $k$  negations, decide  $\llbracket \Phi \rrbracket_{\mathcal{A}} \neq \emptyset$ .

### 3 Representations and parametrised complexity of signatures

Per se, a structure  $\mathcal{A}$  cannot be analysed algorithmically, in particular because the elements of  $A$  do not have a notion of size. A standard way to resolve this issue is defining computability via the notion of representations (as it is done for instance in computable analysis [26]).

**Representations.** A *representation* for a set  $A$  is a surjective partial map  $\rho : \subseteq \Sigma^* \rightarrow A$ , where  $\Sigma$  is a finite alphabet. Words  $w \in \Sigma^*$  are naturally equipped with a notion of *size*, i.e., their length, denoted by  $|w|$ . Not all words are valid representations for elements of  $A$  ( $\rho$  is partial) and each element from  $A$  can be represented in several ways ( $\rho$  is not assumed to be injective). Given a representation  $\rho : \subseteq \Sigma^* \rightarrow A$ , we write  $(\approx_\rho) \subseteq \Sigma^* \times \Sigma^*$  for the equivalence relation  $\{(w_1, w_2) : w_1, w_2 \in \text{dom}(\rho) \text{ and } \rho(w_1) = \rho(w_2)\}$  and define  $h_\rho : \text{dom}(\rho)/\approx_\rho \rightarrow A$  to be the bijection satisfying  $h_\rho([w]_{\approx_\rho}) = \rho(w)$ , for every  $w \in \text{dom}(\rho)$ . Here,  $\text{dom}(\rho)/\approx_\rho$  is the set of all equivalence classes  $[w]_{\approx_\rho}$  of words  $w \in \text{dom}(\rho)$ .

It is often more practical to represent elements of  $A$  by objects that are more sophisticated than words in  $\Sigma^*$ , such as tuples, automata, graphs, etc. Taking these representations does not change the notion of computability or complexity, because they can be easily encoded as words (over a bigger alphabet, if necessary). In our setting, of particular interest are representations as tuples of words. The notion of size for words trivially extends to tuples:  $|(w_1, \dots, w_n)| := n + \sum_{i=1}^n |w_i|$ . Given representations  $\rho : \subseteq \Sigma^* \rightarrow A$  and  $\rho' : \subseteq \Pi^* \rightarrow A'$ , we rely on the following operations on representations:

- The Cartesian product  $\rho \times \rho'$  of representations, defined as in Section 2.
- The representation  $\text{seq}(\rho) : \subseteq \text{seq}(\Sigma^*) \rightarrow \text{seq}(A)$  returning  $(\rho(w_1), \dots, \rho(w_n))$  on tuples  $(w_1, \dots, w_n) \in \text{dom}(\rho)^n$ , for every  $n \in \mathbb{N}$ .
- Given a map  $\oplus : S \times S \rightarrow S$  with  $S \supseteq A$ , the representation  $\text{fold}[\oplus](\rho)$  recursively defined as  $\text{fold}[\oplus](\rho)(()) := \emptyset$ , and  $\text{fold}[\oplus](\rho)((w_1, \dots, w_n)) := \rho(w_1) \oplus \text{fold}[\oplus](\rho)((w_2, \dots, w_n))$ .

We also require representations for basic objects such as  $\mathbb{N}$ ,  $\mathbb{Z}$  and so on. Specifically, we assume to have *canonical representations*  $\nu_X$  for the following countable domains  $X$ :

- $X = \mathbb{N}$  or  $X = \mathbb{Z}$ , so that  $\nu_X$  denote a representation of  $\mathbb{N}$  or  $\mathbb{Z}$ , respectively.
- $X$  is any finite set, e.g., we assume to have a representation  $\nu_{\mathbb{B}}$  for the Booleans  $\mathbb{B} = \{\top, \perp\}$ ,
- $X = \Sigma^*$  where  $\Sigma$  is any finite alphabet. In this case,  $\nu_{\Sigma^*} := \text{id}_{\Sigma^*}$ .

**Implementations and computability.** Let  $\rho : \subseteq \Pi^* \rightarrow A$  and  $\rho_1, \dots, \rho_n$  be representations, with  $\rho_i : \subseteq \Sigma_i^* \rightarrow A_i$ . A function  $f : A_1 \times \dots \times A_n \rightarrow A$  is said to be  $(\rho_1 \times \dots \times \rho_n, \rho)$ -*computable* if there is a computable function (in the usual sense of Turing machines)  $F : \text{dom}(\rho_1) \times \dots \times \text{dom}(\rho_n) \rightarrow \text{dom}(\rho)$  such that  $\rho(F(w_1, \dots, w_n)) = f(\rho_1(w_1), \dots, \rho_n(w_n))$  for all  $w_i \in \text{dom}(\rho_i)$ ,  $i \in [1, n]$ . The function  $F$  is said to be a  $(\rho_1 \times \dots \times \rho_n, \rho)$ -*implementation* of  $f$ . It is convenient to avoid mentioning the representations of a computable function when it operates on canonical types. Given sets  $A, A_1, \dots, A_n$  admitting canonical representations, a function  $f : A_1 \times \dots \times A_n \rightarrow A$  is said to be *computable* whenever it is  $(\nu_{A_1} \times \dots \times \nu_{A_n}, \nu_A)$ -*computable* ( $\nu_{A_i}$  and  $\nu_A$  are the canonical representations of  $A_i$  and  $A$ ).

Let  $\mathcal{A} = (A, \sigma, I)$  be a structure and  $\rho : \subseteq \Sigma^* \rightarrow A$  be a representation. Let  $\mathcal{M} := (\text{dom}(\rho), \sigma, J)$  be a structure where the interpretation function  $J$  associates computable functions to each function, family of functions and relations in  $\sigma$ , and makes  $\approx_\rho$  a congruence for  $\mathcal{M}$ . The structure  $\mathcal{M}$  is said to be a  $\rho$ -*implementation* of  $\mathcal{A}$  whenever  $\rho$  is a homomorphism between  $\mathcal{M}$  and  $\mathcal{A}$ . We highlight the fact that, compared to a standard homomorphism between structures, an implementation is always surjective (since  $\rho$  is surjective) and forces  $J$  to give an interpretation to functions and relations in  $\sigma$  in terms of computable functions.

► **Example 2.** The addition function  $+: \mathbb{Z}^2 \rightarrow \mathbb{Z}$  is  $(\nu_{\mathbb{Z}}^2, \nu_{\mathbb{Z}})$ -computable. Since  $\nu_{\mathbb{Z}}$  is the canonical representation of  $\mathbb{Z}$ , we simply say that  $+$  is *computable*. This is the standard notion of computability over  $\mathbb{Z}$ . The structure  $(\text{dom}(\nu_{\mathbb{Z}}), +)$  is a  $\nu_{\mathbb{Z}}$ -implementation of  $(\mathbb{Z}, +)$ .

**Parametrised complexity of signatures.** The framework we define in the next section requires the introduction of a notion of parametrised complexity for the signature of a structure (which we call a *UXP signature*) which we now formulate. First, let us recall the standard notion of UXP reduction from parametrised complexity theory [11, Chapter 15]. Let  $\Gamma$  and  $\Pi$  be two finite alphabets, and  $D \subseteq \Gamma^*$ . A *parameter function* is a map  $\eta: \Gamma^* \rightarrow \mathbb{N}$  such that  $\eta(w) \geq 1$  for every  $w \in \Gamma^*$ . A computable function  $F: D \rightarrow \Pi^*$  is said to be a *uniform slicewise polynomial reduction* for two parameter functions  $\eta$  and  $\theta$ , or  $(\eta, \theta)$ -UXP reduction in short, whenever there is an increasing map  $G: \mathbb{N} \rightarrow \mathbb{N}$  such that for every  $w \in \Gamma^*$ ,  $F(w)$  runs in time  $|w|^{G(\eta(w))}$  (w.l.o.g. assume  $|w| \geq 2$ ) and  $\theta(F(w)) \leq G(\eta(w))$ .

As usual in computability theory, functions  $F$  with multiple arguments are handled by introducing a special symbol to the alphabet  $\Gamma$ , say  $\#$ , to separate the arguments, thus seeing  $F$  as a function in one input. For instance, an operator  $\oplus: \Sigma_1^* \times \Sigma_2^* \rightarrow \Sigma^*$  can be interpreted by a computable function taking as inputs words  $w_1 \# w_2$  with  $(w_1, w_2) \in \Sigma_1^* \times \Sigma_2^*$ . The product  $(\eta_1 \cdot \eta_2)(w_1 \# w_2) := \eta_1(w_1) \cdot \eta_2(w_2)$  of parameter functions  $\eta_1: \Sigma_1^* \rightarrow \mathbb{N}$  and  $\eta_2: \Sigma_2^* \rightarrow \mathbb{N}$  can be used to refine the complexity analysis of  $\oplus$  to each of its two arguments. We write  $\mathbf{1}$  for the trivial parameter function defined as  $\mathbf{1}(w) := 1$  for all  $w \in \Sigma^*$ .

Let  $\mathcal{A} = (A, \sigma, I)$  be a structure,  $\sigma = (\mathcal{F}, \mathcal{G}, \mathcal{R}, ar)$ ,  $\rho: \subseteq \Sigma^* \rightarrow A$  be a representation, and  $\eta: \Sigma^* \rightarrow \mathbb{N}$  be a parameter function. We say that  $\mathcal{A}$  has a  $(\rho, \eta)$ -UXP signature whenever there is an interpretation function  $J$  such that (i)  $(\text{dom}(\rho), \sigma, J)$  is a  $\rho$ -implementation of  $\mathcal{A}$  and (ii)  $J$  associates a  $(\eta^{ar(f)}, \eta)$ -UXP reduction to every  $f \in \mathcal{F}$ , a  $(\mathbf{1} \cdot \eta^{ar(g)}, \eta)$ -UXP reduction to every  $(g, X) \in \mathcal{G}$ , and a  $(\eta^{ar(R)}, \mathbf{1})$ -UXP reduction to every  $R \in \mathcal{R}$ .

► **Example 3.** Consider the structure  $(L, \cup, \cap, (\cdot)^c)$  where  $L$  is the set of all regular languages over a finite alphabet  $\Sigma$  and  $\cup, \cap$ , and  $(\cdot)^c$  are the canonical operations of union, intersection and complementation of languages, respectively. This structure has a tractable signature for the representation of regular languages as deterministic finite automata (DFAs), as all operations can be implemented in PTIME on DFAs. However, it does not have a tractable signature for the representation of regular languages as non-deterministic finite automata (NFAs), because computing  $(\cdot)^c$  on NFAs requires first to determinise the automaton.

As in the case of representations, it is often more practical to have parameter functions from objects that are more sophisticated than words. Given a parameter function  $\theta: \Sigma^* \rightarrow \mathbb{N}$ , in this paper we consider the following operations  $\text{len}(\theta): \text{seq}(\Sigma^*) \rightarrow \mathbb{N}$ ,  $\text{max}(\theta): \text{seq}(\Sigma^*) \rightarrow \mathbb{N}$  and  $\text{depth}(\theta): \text{seq}(\text{seq}(\Sigma^*)) \rightarrow \mathbb{N}$  on parameter functions (below,  $\mathbf{w} = (w_1, \dots, w_n)$ ):

$$\text{len}(\theta)(\mathbf{w}) := \sum_{i=1}^n \theta(w_i), \quad \text{max}(\theta)(\mathbf{w}) := \max_{i=1}^n \theta(w_i), \quad \text{depth}(\theta) := \text{len}(\text{len}(\theta)).$$

## 4 A framework for the fixed negation fragment of first-order theories

Fix a structure  $\mathcal{A} = (A, \sigma, I)$  and consider  $\text{FO}(\mathcal{A}) := ([\mathcal{A}]_{\text{FO}}, \perp, \top, \vee, \wedge, -, (\pi, \mathbf{I}), (\pi^\forall, \mathbf{I}), \leq)$ . In this section, we describe a framework that can be employed to show that the  $k$  negation satisfiability problem for  $\text{FO}(\mathcal{A})$  is in PTIME. Part of our framework is generic, i.e., applies to any first-order theory, while other parts are necessarily specific to the theory under study. This section covers the former part and highlights the latter.

To understand the framework it is helpful to take a moment to consider how we can exploit the fact that we only consider formulae with a fixed number of negations. For simplicity, let us focus for the time being on *quantified Boolean formulae* (QBF) in prenex

form. A first key question is whether bringing the quantifier-free part  $\Phi$  of a QBF formula in a particular normal form can be computationally beneficial. Of course, due to our restrictions,  $\Phi$  can be brought into DNF in PTIME. However, because of quantifier alternation together with the unbounded number of conjunctions, choosing this normal form comes with several intricacies. Another option we might try is to put  $\Phi$  into a form where all but a fixed amount of constraints are in Horn form, and then try to rely on the algorithm to solve quantified Horn Boolean satisfiability in PTIME [17]. This works for the Boolean case, but not for an arbitrary theory. For instance, the quantified Horn satisfiability problem for the FO theory of  $\mathcal{Z} = (\mathbb{Z}, 0, 1, +, =)$ , i.e. weak Presburger arithmetic, is already NP-hard for the alternation prefix  $\exists\forall$  and 2 variables, as we briefly sketch in Appendix A, and NEXPTIME-hard in general [9]. It turns out that a suitable normal form for  $\Phi$  is given by formulae of the form  $\Phi_1 - (\Phi_2 - (\dots - (\Phi_{k-1} - \Phi_k)))$ , where each  $\Phi_i$  is a negation-free formula in DNF, and  $\Psi_1 - \Psi_2$  is the relative complementation  $\Psi_1 \wedge \neg\Psi_2$ . As we will see in this section, this atypical normal form (introduced by Hausdorff in [13] and called *difference normal form* in [14]) not only fully makes use of our restriction on the number of negations, but also exhibits nice properties in relation to quantification.

A second key question is what representation of  $\llbracket \mathcal{A} \rrbracket_{\text{FO}}$  works best for our purposes, as formulae might not be the right “data structure”. Though the difference normal form already sets how to treat disjunctions and negations, we have the flexibility to vary the representation of conjunctions of atomic formulae. Let us be a bit more precise. Consider a domain  $D \subseteq \llbracket \mathcal{A} \rrbracket_{\text{FO}}$  containing *at least* the sets  $\llbracket \Psi \rrbracket_{\mathcal{A}}$ , for all  $\Psi \in \text{AC}(\sigma)$ . We define  $\text{un}(D)$  to be the closure of  $D$  under the disjunction  $\vee$ , and  $\text{diffnf}(D)$  to be the smallest set containing  $\text{un}(D)$  and being closed under relative complements  $X - Y$ , with  $X \in \text{un}(D)$  and  $Y \in \text{diffnf}(D)$ . From the notion of difference normal form we conclude that  $\{\llbracket \Phi \rrbracket_{\mathcal{A}} : \Phi \text{ quantifier free}\} \subseteq \text{diffnf}(D)$ . Now, given a representation  $\rho : \subseteq \Sigma^* \rightarrow D$ , the partial functions  $\text{un}(\rho) : \subseteq \text{seq}(\Sigma^*) \rightarrow \text{un}(D)$  and  $\text{diffnf}(\rho) : \subseteq \text{seq}(\text{seq}(\Sigma^*)) \rightarrow \text{diffnf}(D)$ , defined as  $\text{un}(\rho) := \text{fold}[\vee](D)$  and  $\text{diffnf}(\rho) := \text{fold}[-](\text{un}(\rho))$  are representations of  $\text{un}(D)$  and  $\text{diffnf}(D)$ , respectively. When  $\rho$  is a representation encoding  $D$  as conjunctions of atomic formulae, then  $\text{diffnf}(\rho)$  is encoding  $\text{diffnf}(D)$  in the difference normal form [13]. The key point is that the representation  $\rho$  can be selected so that  $D$  is encoded as something other than formulae. For instance, for linear arithmetic theories, alternative encodings are given by automata [6] or geometrical objects [10]. In Section 6 we will use the latter. Of course, selecting representations other than formulae requires an efficient way of changing representation, as stressed in the forthcoming Proposition 4 (see the map  $F$ ). One last observation: above, we introduced  $D$  so that it could include sets beyond  $\llbracket \Psi \rrbracket_{\mathcal{A}}$ , where  $\Psi \in \text{AC}(\sigma)$ . Further sets might be required in order to make  $\text{diffnf}(D)$  closed under (universal) projection. For instance, in weak integer arithmetic, the formula  $\exists y : x = 2 \cdot y$ , stating that  $x$  is even, cannot be expressed with a quantifier-free formula, hence  $\llbracket \exists y : x = 2 \cdot y \rrbracket_{\mathcal{Z}}$  must be added to  $D$ .

The following proposition formalises the observations done in the last two paragraphs. We recall that an algorithm is said to be in  $\chi$ -UXP, for a parameter  $\chi : \Sigma^* \rightarrow \mathbb{N}$ , if it runs in time  $|w|^{G(\chi(w))}$  for every  $w \in \Sigma^*$ , for some function  $G : \mathbb{N} \rightarrow \mathbb{N}$  not depending on  $w$ . A decision problem is in  $\chi$ -UXP whenever there is a  $\chi$ -UXP algorithm solving that problem.

► **Proposition 4.** *Fix  $k \in \mathbb{N}$ . Assume the following objects to be defined:*

- a representation  $\rho$  of  $D := \bigcup_{n \in \mathbb{N}} D_n$ , where, for all  $n \in \mathbb{N}$ ,  $D_n \subseteq \mathcal{P}(A^n)$  is such that  $\llbracket \Psi \rrbracket_{\mathcal{A}} \in D_n$  for every  $\Psi \in \text{AC}(\sigma)$  having maximum variable  $x_n$ ,
  - a  $(\xi, \theta)$ -UXP reduction  $F : \text{AC}(\sigma) \rightarrow \text{dom}(\rho)$  s.t.  $(\rho \circ F)(\Psi) = \llbracket \Psi \rrbracket_{\mathcal{A}}$  for all  $\Psi \in \text{AC}(\sigma)$ .
- If  $\mathcal{D} = (\text{diffnf}(D), \perp, \top, \vee, \wedge, -, (\pi, \mathbf{I}), (\pi^\forall, \mathbf{I}), \leq)$  has  $(\text{diffnf}(\rho), \text{depth}(\theta))$ -UXP signature,
- the  $k$  negations satisfiability problem for  $\text{FO}(\mathcal{A})$  is in  $\xi$ -UXP (PTIME, if  $\xi = \mathbf{1}$ ), and
  - there is a  $\xi$ -UXP (PTIME, if  $\xi = \mathbf{1}$ ) algorithm that, given a formula  $\Phi$  of  $\text{FO}(\mathcal{A})$  having at most  $k$  negations, returns  $X$  in  $\text{dom}(\text{diffnf}(\rho))$  such that  $\text{diffnf}(\rho)(X) = \llbracket \Phi \rrbracket_{\mathcal{A}}$ .

By virtue of what we said above, Proposition 4 should not be surprising: the reduction  $F$  enables an efficient conversion from  $\text{AC}(\sigma)$  to elements in  $\text{dom}(\rho)$ , and (since  $\mathcal{D}$  is a structure)  $\text{diffnf}(\mathcal{D})$  is closed under all the operations in the signature and thus it is equal to  $\llbracket \mathcal{A} \rrbracket_{\text{FO}}$ . Consequently,  $\text{FO}(\mathcal{A})$  has a  $(\text{diffnf}(\rho), \text{depth}(\theta))$ -UXP signature, and one can efficiently use  $\text{diffnf}(\rho)$  as a data structure to carry out the algorithm to decide satisfiability, by simply invoking the various UXP reductions implementing the functions and relations in  $\mathcal{D}$ . We remark that the sole purpose of the parameter  $\theta$  is to factor in the parameter  $\xi$ , and can be thought as  $\theta := \mathbf{1}$  in the case of  $\xi := \mathbf{1}$  (i.e., the case yielding PTIME algorithms).

Whereas the choice of  $\mathcal{D}$  and  $F$  highly depends on the FO theory at hand, we show that a significant portion of the work required to prove that  $\mathcal{D}$  has the desired UXP signature can be treated in a general way, thanks to the notion of difference normal form. This “automation” can be seen as the core of our framework, which provides a minimal set of subproblems that are sufficient to conclude that  $\mathcal{D}$  has a  $(\text{diffnf}(\rho), \text{depth}(\theta))$ -UXP signature. Below, we divide those subproblems into two steps, one for Boolean connectives and one for quantification. One significant result in this context is that negation can be treated in a general way.

► **Step 1** (Boolean connectives).

1. Show that the structure  $(\mathcal{D}, \wedge, \leq)$  has a  $(\rho, \theta)$ -UXP signature.
2. Show that the structure  $(\text{un}(\mathcal{D}), \leq)$  has a  $(\text{un}(\rho), \text{len}(\theta))$ -UXP signature.

Step 1 asks to provide algorithms for solving typical computational problems that are highly domain-specific: Item 1 considers the intersection and inclusion problems for elements of  $\mathcal{D}$ , with respect to the representation  $\rho$ , whereas Item 2 deals with the inclusion problem for unions of elements in  $\mathcal{D}$ , with respect to the representation  $\text{un}(\rho)$ . In the case of unions, we highlight the parameter  $\text{len}(\theta)$  which fixes the length of the union.

Once Step 1 is established, we are able to show that the full Boolean algebra (including relative complementation) of  $\text{diffnf}(\mathcal{D})$  has a UXP signature that is suitable for Proposition 4.

► **Lemma 5.** *Under the assumption that Step 1 is established,  $(\text{diffnf}(\mathcal{D}), \perp, \top, \vee, \wedge, -, \leq)$  has a  $(\text{diffnf}(\rho), \text{depth}(\theta))$ -UXP signature.*

The proof of this lemma boils down to the definition of suitable UXP reductions implementing the binary operations  $\wedge$ ,  $\vee$  and  $-$ . We will give further insights on how this is achieved in Section 5, where we study algorithmic aspects of the difference normal form.

Moving forward, we now consider projection and universal projection. Again, the goal is to minimise the efforts needed to add support for these operations. In this sense, the decision to adopt the difference normal becomes now crucial. First, we need to introduce a variant of the universal projection which we call relative universal projection. Given  $Z \subseteq A^m$ ,  $X \subseteq A^n$  and  $\mathbf{i} = (i_1, \dots, i_k) \in \mathbf{I}$ , the *relative universal projection*  $\pi_Z^\forall(\mathbf{i}, X)$  of  $X$  with respect to  $Z$  is defined as follows (where  $M := \max(m, n)$ ):

$$\pi_Z^\forall(\mathbf{i}, X) := \{(a_1, \dots, a_M) \in A^M : \mathbf{a} := (a_1, \dots, a_m) \in \pi(\mathbf{i}, Z) \text{ and for all } \mathbf{b} \in A^k \\ \text{if } \mathbf{a}[\mathbf{i} \leftarrow \mathbf{b}] \in Z \text{ then } (a_1, \dots, a_n)[\mathbf{i} \leftarrow \mathbf{b}] \in X\}.$$

Informally speaking,  $\pi_Z^\forall(\mathbf{i}, X)$  acts as a universal projection for the part of  $X$  that lies inside  $Z$ . Note that one can retrieve the universal projection as  $\pi^\forall(\mathbf{i}, X) = \pi_Z^\forall(\mathbf{i}, X)$ .

The lemma below outlines a key “mutual distribution” property of projection and relative universal projection over relative complement. In the context of the difference normal form, this property allows us to disregard complementation when adding support for quantification.

► **Lemma 6.** *We have  $\pi(\mathbf{i}, X - Y) = \pi(\mathbf{i}, X) - \pi_X^\forall(\mathbf{i}, Y)$  and  $\pi_Z^\forall(\mathbf{i}, X - Y) = \pi_Z^\forall(\mathbf{i}, X) - \pi(\mathbf{i}, Y)$ , for every  $X \subseteq A^n$ ,  $Y \subseteq A^m$ ,  $Z \subseteq A^r$ , and  $\mathbf{i} \in \mathbf{I}$ .*

For instance,  $\pi(\mathbf{i}, W - (X - (Y - Z))) = \pi(\mathbf{i}, W) - (\pi_W^\forall(\mathbf{i}, X) - (\pi(\mathbf{i}, Y) - \pi_Y^\forall(\mathbf{i}, Z)))$ .

Below, let us write  $\dot{\pi}$  for the restriction of the projection operator  $\pi$  on inputs  $(\mathbf{i}, X)$  where  $X \in \mathbf{D}$ , and write  $\dot{\pi}^{\forall}$  for the restriction of the relativised universal projection  $\pi^{\forall}$  on inputs  $(Z, \mathbf{i}, X)$  where  $Z \in \overline{\mathbf{D}}$  and  $X \in \text{un}(\mathbf{D})$ . Thanks to Lemma 6, adding to Step 1 the following step is sufficient to conclude that  $\mathcal{D}$  has the UXP signature required by Proposition 4.

► **Step 2** (Projection and universal projection).

1. Show that  $\dot{\pi}(\mathbf{i}, X) \in \text{diffnf}(\mathbf{D})$ , for every  $X \in \mathbf{D}$  and  $\mathbf{i} \in \mathbf{I}$ .
2. Show that  $\dot{\pi}_Z^{\forall}(\mathbf{i}, X) \in \text{diffnf}(\mathbf{D})$ , for every  $Z \in \mathbf{D}$ ,  $\mathbf{i} \in \mathbf{I}$  and  $X \in \text{un}(\mathbf{D})$ .
3. Show a  $(\mathbf{1} \cdot \theta, \text{depth}(\theta))$ -UXP reduction that  $(\nu_{\mathbf{I}} \times \rho, \text{diffnf}(\rho))$ -implements  $\dot{\pi}$ .
4. Show a  $(\theta \cdot \mathbf{1} \cdot \text{len}(\theta), \text{depth}(\theta))$ -UXP reduction that  $(\rho \times \nu_{\mathbf{I}} \times \text{un}(\rho), \text{diffnf}(\rho))$ -implements  $\dot{\pi}^{\forall}$ .

► **Lemma 7.** Under the assumption that Steps 1 and 2 are established, the structure  $\mathcal{D}$  from Proposition 4 has a  $(\text{diffnf}(\rho), \text{depth}(\theta))$ -UXP signature.

## 5 Closing prelattices under relative complement

Let us go back to the notion of difference normal form, which again are formulae of the form  $\Phi_1 - (\Phi_2 - (\dots - \Phi_k))$ , where each  $\Phi_i$  is negation-free and in DNF. Our framework is based on the idea of using these syntactic chains of relative complementations, let us call them *decreasing sequences*, as a way of closing a structure (not just formulae) under complement. Doing this allows the domain  $\mathbf{D}$  to not be necessarily closed under complement. Then, one natural question to ask is under which assumptions do structures admit a computable notion of decreasing sequences. We give an answer to this question by showing that any prelattice  $\mathcal{A}$  that is well-founded *or* distributive has a well-defined algebra over decreasing sequences (SDS algebra). We study computational aspects of SDS algebras that allows us to establish Lemma 5. When  $\mathcal{A}$  is both well-founded and distributive, we also show that the SDS algebra act as a completion of  $\mathcal{A}$  under relative complement (this result is not required for Lemma 5).

**Prelattices.** We assume familiarity with the order theoretic definition of a *lattice*. A structure  $\mathcal{A} = (A, \vee, \wedge, \leq)$  is said to be a *prelattice* whenever  $\leq$  is a *preorder*, and the quotient structure  $\mathcal{A}/\approx := (A/\approx, \vee/\approx, \wedge/\approx, \leq/\approx)$  of  $\mathcal{A}$  under the congruence  $(\approx) := (\leq \cap \leq^{-1})$  is a lattice. Roughly speaking, a prelattice is a lattice that may not satisfy antisymmetry, i.e., distinct elements of  $A$  are allowed to be equal under  $\leq$ . If  $\leq/\approx$  has a least element  $L$ , we often add to the signature of  $\mathcal{A}$  a constant symbol  $\perp$  interpreted as an element  $a \in A$  such that  $[a]_{\approx} = L$ . As usual,  $\mathcal{A}$  is *well-founded* if there are no infinite sequences of elements that are strictly decreasing with respect to  $\leq$ , and it is *distributive* whenever  $a \wedge (b \vee c) \approx (a \wedge b) \vee (a \wedge c)$ , for every  $a, b, c \in A$ . A (pre)lattice  $(L, \leq)$  is said to be *closed under relative complement* whenever, for every  $x, y \in L$  there is an element  $[x \setminus y] \in L$  satisfying  $y \wedge [x \setminus y] = \perp$  and  $x \leq y \vee [x \setminus y]$ . (Pre)lattices may not be closed under relative complement, see for example the three elements lattice  $(\{\perp, p, \top\}, \leq)$  with  $\perp < p < \top$ .

**Decreasing sequences and SDS algebras.** Fix a prelattice  $\mathcal{A} = (A, \perp, \vee, \wedge, \leq)$  that is well-founded or distributive. Let  $\approx$  and  $<$  be the equivalence relation and strict partial order induced by  $\leq$ , respectively. We write  $\text{SDS}(\mathcal{A})$  for the set of all *strictly decreasing sequences* (SDS) over  $A$ , i.e., those (possibly empty) finite tuples  $(a_1, \dots, a_n) \in \text{seq}(A)$  satisfying  $a_{i+1} < a_i$  for every  $i \in [1, n-1]$ . Let  $X := (a_1, \dots, a_n), Y := (b_1, \dots, b_m) \in \text{SDS}(\mathcal{A})$ . Given  $a \in A$  and  $X' \in \text{SDS}(\mathcal{A})$ , we write  $X = \langle a; X' \rangle$  whenever  $a = a_1$  and  $X' = (a_2, \dots, a_n)$ . We write  $X \approx Y$  whenever  $n = m$  and  $a_i \approx b_i$  for all  $i \in [1, n]$ , or  $X, Y \in \{(), a : a \in A \text{ and } a \approx \perp\}$ . We recursively define the *cons* operator  $(:): A \times \text{SDS}(\mathcal{A}) \rightarrow \text{SDS}(\mathcal{A})$  as follows:

$$a : X := \begin{cases} a & \text{if } X \in \{(), b : b \in A \text{ and } b \approx \perp\} \\ (a, a_1, \dots, a_n) & \text{else if } a_1 < a \\ a : ((a \wedge a_1) : (a_2, \dots, a_n)) & \text{else if } a \wedge a_1 < a \\ \perp & \text{else if } n = 1 \text{ (here, } a \leq a_1) \\ (a \wedge a_2) : (a_3, \dots, a_n) & \text{otherwise} \end{cases}$$

Intuitively, on input  $(a, X)$ ,  $(:)$  returns an SDS that represents the relative complement of  $a$  and  $X$ . With a simple induction on the length of  $X$  one can show that  $(:)$  is well-defined.

The *SDS algebra* of  $\mathcal{A}$  is the structure  $(\text{SDS}(\mathcal{A}), \emptyset, \gamma, \lambda, -, \preceq)$ . In this structure,  $\emptyset$  is the constant function returning  $\perp^{\mathcal{A}}$  from  $\mathcal{A} \subseteq \text{SDS}(\mathcal{A})$ , the (*inclusion*)  $\preceq$  is defined as  $X \preceq Y \stackrel{\text{def}}{=} (X - Y) \approx \emptyset$ , and the functions  $\gamma$  (*union*),  $\lambda$  (*intersection*) and  $-$  (*difference*), having arity two, are interpreted following the mutually recursive definitions (where, whenever their length is non-zero, we assume  $X = \langle a; X' \rangle$  and  $Y = \langle b; Y' \rangle$ ):

$$\begin{aligned} X \wedge Y &:= \begin{cases} \emptyset & \text{if } X \approx \emptyset \text{ or } Y \approx \emptyset, \\ (a \wedge b) : (X' \gamma Y') & \text{else} \end{cases} & X \gamma Y &:= \begin{cases} Y & \text{if } X \approx \emptyset, \\ X & \text{else if } Y \approx \emptyset, \\ a : (X' - Y) & \text{else if } b \leq a, \\ (a \vee b) : ((X' \gamma Y') - ((a \wedge b) : (X' \lambda Y'))) & \text{else.} \end{cases} \\ X - Y &:= \begin{cases} X & \text{if } X \approx \emptyset \text{ or } Y \approx \emptyset, \\ a : (X' \gamma Y') & \text{else} \end{cases} \end{aligned}$$

The definitions of  $\lambda$ ,  $\gamma$  and  $-$  observe validities of elementary set theories, and follow the idea that  $\langle a; X' \rangle$  should represent the element  $a - X'$ . For instance, the last line in the definition of  $X - Y$ , where  $X = \langle a; X' \rangle$ , relies on the set validity  $(E - F) - G = E - (F \cup G)$ .

► **Lemma 8.** *Suppose that  $\mathcal{A}$  is well-founded or distributive. Then, the functions  $\gamma, \lambda, - : \text{SDS}(\mathcal{A}) \times \text{SDS}(\mathcal{A}) \rightarrow \text{SDS}(\mathcal{A})$  and  $\preceq : \text{SDS}(\mathcal{A}) \times \text{SDS}(\mathcal{A}) \rightarrow \mathbb{B}$  are well-defined.*

For  $\mathcal{A}$  well-founded, this lemma is proven by induction on the lexicographic ordering built from  $\leq^{\mathcal{A}}$  and the total ordering on  $\mathbb{N}$ . For  $\mathcal{A}$  distributive, one notes that the closure  $C$  of a finite set of elements  $\{\perp, a_1, \dots, a_n\} \subseteq A$  under  $\vee$  and  $\wedge$  is a prelattice that is finite up to  $\approx$ , making  $\leq^{\mathcal{A}}/\approx$  a well-founded relation when restricted to elements in  $C$ . These elements are the only ones computed by  $\gamma$ ,  $\lambda$  and  $-$ . The lemma then follows as in the well-founded case.

The following proposition clarifies the intention behind SDS algebras.

► **Proposition 9.** *Any well-founded and distributive lattice  $\mathcal{A}$  embeds in  $(\text{SDS}(\mathcal{A}), \emptyset, \gamma, \lambda, \preceq)$ , which is a well-founded distributive prelattice closed under relative complement.*

Showing that  $\mathcal{A}$  embeds in  $(\text{SDS}(\mathcal{A}), \emptyset, \gamma, \lambda, \preceq)$  is simple. To show that the latter structure is closed under relative complement we rely on Birkhoff's representation theorem, a theorem that allows to construct set algebras isomorphic to  $\mathcal{A}$ . Birkhoff's representation theorem is usually given for *finite* distributive lattices. However, an inspection of its proof shows that finiteness can be replaced with well-foundedness in a simple way.

► **Theorem 10** ([3]). *Let  $\mathcal{A} = (A, \perp, \vee, \wedge, \leq)$  be a well-founded distributive lattice. There is an algebra of sets  $\mathcal{B} = (B, \emptyset, \cup, \cap, \subseteq)$  that is isomorphic to  $\mathcal{A}$ .*

Let  $\mathcal{N} = (N, \emptyset, \cup, \cap, \setminus, \subseteq)$  be the algebra of sets obtained by closing the structure  $\mathcal{B}$  above under the set difference  $\setminus$ . Thanks to Theorem 10, to show the closure under relative complement required by Proposition 9 it suffices providing a surjective homomorphism from the SDS algebra of  $\mathcal{B}$  to the structure  $\mathcal{N}$ . The map  $h : \text{SDS}(\mathcal{B}) \rightarrow N$  defined as  $h(X) := \emptyset$  for  $X \approx \emptyset$ , and otherwise as  $h(X) := a \setminus h(X')$  for  $X = \langle a; X' \rangle$ , is such a homomorphism.

**UXP signatures for SDS algebras.** We move to the computational aspects of SDS algebras. Let  $\mathcal{A} = (A, \perp, \vee, \wedge, \leq)$  be a distributive (not necessarily well-founded) prelattice and  $\rho : \subseteq \Sigma^* \rightarrow A$  be a representation. Let  $\text{SDS}(\rho) : \subseteq \text{seq}(\Sigma^*) \rightarrow \text{SDS}(\mathcal{A})$  be the representation of  $\text{SDS}(\mathcal{A})$  defined as  $\text{SDS}(\rho)(w_1, \dots, w_n) := \rho(w_1) : (\dots : (\rho(w_n)))$  for every  $n \in \mathbb{N}$  and  $w_1, \dots, w_n \in \text{dom}(\rho)$ , and undefined otherwise. Below,  $\text{len}(\eta)$  is defined as in Section 3.

► **Theorem 11.** *Let  $\mathcal{A} = (A, \perp, \vee, \wedge, \leq)$  be a distributive prelattice with a  $(\rho, \eta)$ -UXP signature. The SDS algebra of  $\mathcal{A}$  has a  $(\text{SDS}(\rho), \text{len}(\eta))$ -UXP signature.*

For the proof of this theorem, one considers the  $\rho$ -implementation  $\mathcal{R}$  of  $\mathcal{A}$  in which the functions  $\vee$  and  $\wedge$  are  $(\eta \cdot \eta, \eta)$ -UXP reductions and the relation  $\leq$  is a  $(\eta \cdot \eta, \mathbf{1})$ -UXP reduction. The structure  $\mathcal{R}$  is a distributive prelattice, hence it has a well-defined SDS algebra  $(\text{SDS}(\mathcal{R}), \emptyset, \Upsilon, \lambda, -, \preceq)$ . By following the above definitions of  $\Upsilon, \lambda, -$  and  $\preceq$ , one can provide  $(\text{len}(\eta)^2, \text{len}(\eta))$ -UXP and  $(\text{len}(\eta) \cdot \text{len}(\eta), \mathbf{1})$ -UXP reductions for the functions and relations of the structure  $\mathcal{S} = (\text{dom}(\text{SDS}(\rho)), \emptyset, \Upsilon, \lambda, -, \preceq)$  that  $\text{SDS}(\rho)$ -implements the SDS algebra of  $\mathcal{A}$ .

Theorem 11 gives us what we need to prove Lemma 5. Step 1 of the framework implies that  $\mathcal{U} = (\text{un}(\mathcal{D}), \perp, \vee, \wedge, \leq)$  is a distributive prelattice with a  $(\text{un}(\rho), \text{len}(\theta))$ -UXP signature. By Theorem 11, the SDS algebra of  $\mathcal{U}$  has a  $(\text{SDS}(\text{un}(\rho)), \text{depth}(\theta))$ -UXP signature. So, Lemma 5 follows as one provides a surjective homomorphism from the SDS algebra of  $\mathcal{U}$  to  $(\text{diffnf}(\mathcal{D}), \perp, \top, \vee, \wedge, -, \preceq)$ . This surjective homomorphism is obtained by updating map  $h$  used in the proof of Proposition 9 so that it uses the operator  $-$  instead of the set difference  $\setminus$ .

## 6 Weak linear integer arithmetic

In this section, we briefly discuss how to instantiate the framework of Section 4 to weak Presburger arithmetic (weak PA), i.e. the FO theory of the structure  $\mathcal{Z} = (\mathbb{Z}, 0, 1, +, =)$ .

**Setup.** According to Proposition 4, instantiating the framework requires first to define the domain  $\mathcal{D}$ , its representation  $\rho$  and the change of representation  $F : \text{AC}(\sigma) \rightarrow \text{dom}(\rho)$ . In weak PA, conjunctions of atomic formulae are systems of affine equations, which over  $\mathbb{Z}$  define shifted integer lattices (SL), which are not necessarily fully dimensional. We let  $\mathcal{D}_n$  be the set of all shifted (integer) lattices of  $\mathbb{Z}^n$ , so that  $\mathcal{D}$  is the set of all shifted lattices of  $\mathbb{Z}^n$  for some  $n$ . We represent elements in  $\mathcal{D}$  with the standard representation of a SL as a base point and an independent periodic set. Recall that we write  $\nu_{\mathbb{Z}^n}$  for the canonical representation of  $\mathbb{Z}^n$  (see Section 3). Formally, for every  $n \in \mathbb{N}$ , if  $v_0$  represents a vector in  $\mathbb{Z}^n$ , and  $v_1, \dots, v_k$  represent linearly independent vectors in  $\mathbb{Z}^n$ , then we let

$$\rho_{\text{SL}}(n, v_0, \dots, v_k) := \nu_{\mathbb{Z}^n}(v_0) + \text{span}_{\mathbb{Z}}\{\nu_{\mathbb{Z}^n}(v_1), \dots, \nu_{\mathbb{Z}^n}(v_k)\}.$$

A PTIME function  $F$  allowing to change representation from conjunctions of atomic formulae to elements in  $\text{dom}(\rho_{\text{SL}})$  can be obtained thanks to the following well-known algorithm.

► **Proposition 12** ([16]). *There is a PTIME algorithm to compute the Hermite normal form, along with the transformation matrices, of a given matrix with integer entries.*

Since  $F$  runs in PTIME, the parameter  $\xi$  in Proposition 4 equals  $\mathbf{1}$ , and we need to show that  $\mathcal{D}$  has a  $(\text{diffnf}(\rho_{\text{SL}}), \text{depth}(\theta))$ -UXP signature for  $\theta := \mathbf{1}$ , by establishing Steps 1 and 2.

**Step 1.** Both the problems of computing intersections and testing inclusions for two shifted lattices represented as in  $\rho_{\text{SL}}$  reduces to solving systems of linear equations over  $\mathbb{Z}$ , which can be done in PTIME again thanks to Proposition 12. This establishes Item 1 of Step 1.

Item 2 asks for an algorithm to solve inclusion between union of shifted lattices. This is a computationally expensive operation. Roughly speaking, one first notices that it suffices to be able to test  $Z \leq X$  where  $Z \subseteq \mathbb{Z}^k$  is a SL and  $X := \bigvee_{i \in I} X_i \subseteq \mathbb{Z}^k$ , where all  $X_i$  are SL. This inclusion testing can be performed by checking that  $|Z \cap [0, d)^k| = |(Z \wedge X) \cap [0, d)^k|$  for a well-chosen value of  $d \in \mathbb{N}$ . Hence, inclusion reduces to a counting problem for lattice points in a box, which in turn reduces to computing lattice determinants:

► **Lemma 13.** *Let  $L \subseteq \mathbb{Z}^k$  be a lattice and  $d > 0$  such that  $d\mathbb{Z}^k \subseteq L$ , then  $|L \cap [0, d)^k| = \frac{d^k}{\det(L)}$ .*

To extend the above lemma to union of shifted lattices, we rely on an inclusion-exclusion formula which requires the algorithm to consider all possible  $2^{|I|}$  intersections between the  $X_i$ . This leads to a procedure that is exponential in the number of elements in the union, which is however sufficient to establish Item 2, since it asks for an algorithm that runs in PTIME when the length of the union is fixed; see the parameter  $\text{len}(\theta)$ .

► **Lemma 14.** *There is an algorithm that given  $z \in \text{dom}(\rho_{\text{SL}})$  and  $x \in \text{dom}(\text{un}(\rho_{\text{SL}}))$  decides  $\rho_{\text{SL}}(z) \leq \text{un}(\rho_{\text{SL}})(x)$  in time  $2^{\text{len}(\mathbf{1})(x)} \text{poly}(|x|, |z|)$ . In particular, Item 2 of Step 1 holds.*

**Step 2.** Establishing Items 1 and 3 is simple: thanks to our choice of representation, i.e.  $\rho_{\text{SL}}$ , given  $X \in \text{dom}(\rho_{\text{SL}})$  and  $\mathbf{i} \in \mathbf{I}$ ,  $\pi(\mathbf{i}, X)$  can be computed by simply crossing out the entries of all vectors of  $X$  corresponding to the indices in  $\mathbf{i}$ . On the contrary, the algorithm for universal projections  $\pi_{\mathbb{Z}}^{\forall} X$  required by Items 2 and 4 turns out to be challenging to compute. Intuitively, similarly to inclusion testing, we need to count points in unions of SL but in a parametric way. This means that given  $X = \bigvee_{\ell \in L} X_{\ell}$ , where all  $X_{\ell}$  are SL, every intersection  $\bigwedge_{j \in J} X_j$  ( $J \subseteq L$ ) in the inclusion-exclusion formula may or may not need to be counted, depending on the value of a parameter  $f$  belonging of a certain set of parameters  $\mathcal{F}$  (see lemma below). The algorithm therefore considers all possible ways in which intersections may or may not be taken, which is roughly  $2^{2^{|L|}}$ . This allows us to conclude a rather surprising fact: the relative universal projection can be expressed as a complex combination of unions, intersections, projections and relative complementations that are exclusively applied to the initial sets in input. The number of these operations only depend on the length  $|L|$ .

► **Lemma 15.** *Consider  $X = \bigvee_{\ell \in L} X_{\ell}$ , where  $X_{\ell} \in \text{D}$  for all  $\ell \in L$ , and let  $Z \in \text{D}$  and  $\mathbf{i} \in \mathbf{I}$ . Then, there is a set of parameters  $\mathcal{F} \subseteq (2^L \rightarrow \{0, 1\})$  such that*

$$\pi_{\mathbb{Z}}^{\forall}(\mathbf{i}, X) = \bigvee_{f \in \mathcal{F}} \left( \left( \bigwedge_{J: f(J)=1} \bigwedge_{j \in J} \pi(\mathbf{i}, X_j \wedge Z) \right) - \left( \bigvee_{J: f(J)=0} \bigwedge_{j \in J} \pi(\mathbf{i}, X_j \wedge Z) \right) \right).$$

*Given  $z, (x_{\ell})_{\ell \in L} \in \text{dom}(\rho_{\text{SL}})$  s.t.  $\rho_{\text{SL}}(z) = Z$  and  $\rho_{\text{SL}}(x_{\ell}) = X_{\ell}$ , the set  $\mathcal{F}$  can be computed in time  $\text{poly}(|z|, \max_{\ell \in L}(|x_{\ell}|), 2^{2^{|L|} + |L|})$ . In particular, Items 2 and 4 of Step 2 hold.*

To clarify, since the parameter  $\text{len}(\theta)$  in Item 4 fixes the length  $|L|$ , the right-hand side of the above equation only has a fixed number of operations, and can thus be evaluate efficiently thanks to the other steps of the framework. Then, by Lemma 7 and Proposition 4, we get:

► **Theorem 16.** *Fix  $k \in \mathbb{N}$ . The  $k$  negations satisfiability problems for weak PA is in PTIME.*

By Proposition 4 we also conclude that there is a PTIME procedure that given a formula  $\Phi$  from  $\text{FO}(\mathcal{Z})$  with  $k$  negations returns an element of  $\text{dom}(\text{diff}(\text{un}(\rho_{\text{SL}})))$  representing  $\llbracket \Phi \rrbracket_{\mathcal{Z}}$ .

## 7 Final remarks

We developed a framework to establish polynomial-time decidability of fixed negation sentences of first-order theories whose signatures enjoy certain (parametrised) complexity properties. A key feature of the framework is that it treats complementation in a general way, and considers universal projection as a first-class citizen. Note that, a priori, the latter operation might be easier than the former to decide, as shown for instance in [8].

We instantiated our framework to show that the fixed negation fragment of weak PA is in PTIME, in sharp contrast with (standard) PA [20]. Due to space constraints, we did not provide further instantiation of our framework. We know that it can be used to show that the fixed negation fragment of weak linear real arithmetic is in PTIME. We believe that the framework provides a sensible approach to study fixed negation fragments of FO extensions of, e.g., certain abstract domains. While the framework obviously works for interval arithmetic, the case of linear octagon arithmetic [19], whose full FO theory is PSPACE-complete [2], seems already non-trivial. More generally, since the various steps required to instantiate the framework only consider natural computational problems (inclusions and projections), our hope is to also tackle theories outside the world of arithmetic.

---

## References

- 1 Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Vol 1: Efficient Algorithms*. Foundations of Computing. MIT Press, 1996.
- 2 Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The complexity of Presburger arithmetic with power or powers. In *ICALP, 2023*. To appear.
- 3 Garrett Birkhoff. Rings of sets. *Duke Math. J.*, 3(3):443–454, 1937. doi:10.1215/S0012-7094-37-00334-X.
- 4 Manuel Bodirsky, Barnaby Martin, Marcello Mamino, and Antoine Mottet. The complexity of disjunctive linear diophantine constraints. In *MFCS*, pages 33:1–33:16, 2018. doi:10.4230/LIPIcs.MFCS.2018.33.
- 5 Itshak Borosh and Leon B. Treybing. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Am. Math. Soc.*, 55:299–304, 1976. doi:10.2307/2041711.
- 6 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Math. Logic Quart.*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.
- 7 Hubie Chen. A rendezvous of logic, complexity, and algebra. *ACM Comput. Surv.*, 42(1):2:1–2:32, 2009. doi:10.1145/1592451.1592453.
- 8 Dmitry Chistikov and Christoph Haase. On the complexity of quantified integer programming. In *ICALP*, pages 94:1–94:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.94.
- 9 Dmitry Chistikov, Christoph Haase, Zahra Hadizadeh, and Alessio Mansutti. Higher-order quantified Boolean satisfiability. In *MFCS*, pages 33:1–33:15, 2022. doi:10.4230/LIPIcs.MFCS.2022.33.
- 10 Dmitry Chistikov, Christoph Haase, and Alessio Mansutti. Geometric decision procedures and the VC dimension of linear arithmetic theories. In *LICS*, pages 59:1–59:13, 2022. doi:10.1145/3531130.3533372.
- 11 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 12 Erich Grädel. Simple sentences that are hard to decide. *Inf. Comput.*, 94(1):62–82, 1991. doi:10.1016/0890-5401(91)90033-X.
- 13 Felix Hausdorff. *Grundzüge der Mengenlehre*. Veit and Company, Leipzig, 1914. [Accessed 7th August 2023]. URL: <https://archive.org/details/grundzgedermen00hausuoft>.
- 14 Markus Junker. A note on equational theories. *J. Symb. Log.*, 65(4):1705–1712, 2000. doi:10.2307/2695070.
- 15 Ravindran Kannan. Test sets for integer programs,  $\forall\exists$  sentences. *Polyhedral Combinatorics, Proc. of a DIMACS workshop*, pages 38–48, 1990.

- 16 Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8(4):499–507, 1979. doi:10.1137/0208040.
- 17 Marek Karpinski, Hans Kleine Büning, and Peter H. Schmitt. On the computational complexity of quantified horn clauses. In *CSL*, pages 129–137, 1987. doi:10.1007/3-540-50241-6\_34.
- 18 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.
- 19 Antoine Miné. The octagon abstract domain. *High. Order Symb. Comput.*, 19(1):31–100, 2006. doi:10.1007/s10990-006-8609-1.
- 20 Danny Nguyen and Igor Pak. Short Presburger arithmetic is hard. *SIAM J. Comput.*, 51(2):17:1–30, 2022. doi:10.1137/17M1151146.
- 21 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
- 22 Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. Am. Math. Soc.*, 284:203–218, 1984. doi:10.2307/1999283.
- 23 Uwe Schöning. Complexity of presburger arithmetic with fixed quantifier dimension. *Theory Comput. Syst.*, 30(4):423–428, 1997. doi:10.1007/s002240000059.
- 24 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.
- 25 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. Am. Math. Soc.*, 72(1):155–158, 1978. doi:10.1080/00029890.1978.11994639.
- 26 Klaus Weihrauch. *Computable Analysis*. Springer, 2000. doi:10.1007/978-3-642-56999-9.

## A Hardness of quantified weak PA Horn formulas

In this appendix, we complement the tractability result for the  $k$ -negation fragment of weak PA established in Section 6 with an NP lower bound for (quantified) weak PA Horn formulas. This lower bound only requires two variables  $x$  and  $y$ , where  $x$  is quantified existentially and  $y$  is quantified universally. The matrix of such a 2-variable formula is of the form

$$\bigwedge_{1 \leq i \leq n} (a_i \cdot x + b_i \cdot y = c_i \rightarrow a'_i \cdot x + b'_i \cdot y = c'_i).$$

We show this result by a reduction from the problem of deciding a univariate system of non-congruences  $\bigwedge_{i=1}^k x \not\equiv r_i \pmod{m_i}$ , where  $m_i \geq 2$  and  $r_i \in [0, m_i - 1]$  for every  $i \in [1, k]$ . This problem is shown NP-hard in [1, Theorem 5.5.7]. The reduction directly follows from the following equivalence: for every  $x \in \mathbb{Z}$ ,

$$\bigwedge_{i=1}^k x \not\equiv r_i \pmod{m_i} \iff \forall y : \bigwedge_{i=1}^k (x - r_i = m_i \cdot y \rightarrow y = 3 \cdot x + 1).$$

First, consider  $x \in \mathbb{Z}$  satisfying the left-hand side. Pick  $y \in \mathbb{Z}$ . We have  $x - r_i \neq m_i \cdot y$  for every  $i \in [1, k]$ . So, every antecedent of the implications in  $\bigwedge_{i=1}^k (x - r_i = m_i \cdot y \rightarrow y = 3 \cdot x + 1)$  is false, showing the right-hand side. For the other direction, consider an  $x \in \mathbb{Z}$  satisfying the right-hand side. It suffices to show that, for every  $i \in [1, k]$  and  $y \in \mathbb{Z}$ , if  $x - r_i = m_i \cdot y$  then  $y \neq 3 \cdot x + 1$ . This implies that, for the right-hand side to hold, it must be the case that  $x - r_i \neq m_i \cdot y$  for every  $i \in [1, k]$  and  $y \in \mathbb{Z}$ ; proving the left-hand side. *Ad absurdum*, suppose that  $x - r_i = m_i \cdot y$  and  $y = 3 \cdot x + 1$  hold. Then,  $x = -\frac{r_i + m_i}{3 \cdot m_i - 1}$ . However,  $0 < \frac{r_i + m_i}{3 \cdot m_i - 1} \leq \frac{2 \cdot m_i - 1}{3 \cdot m_i - 1} < 1$ , as  $m_i \geq 2$  and  $r_i \in [0, m_i - 1]$ , contradicting that  $x$  is an integer.

► **Proposition 17.** *Deciding  $\exists \forall$  weak PA Horn sentences in two variables is NP-hard.*

# The Covering Canadian Traveller Problem Revisited

Niklas Hahn  

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Michalis Xeferis  

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

---

## Abstract

In this paper, we consider the  $k$ -Covering Canadian Traveller Problem ( $k$ -CCTP), which can be seen as a variant of the Travelling Salesperson Problem. The goal of  $k$ -CCTP is finding the shortest tour for a traveller to visit a set of locations in a given graph and return to the origin. Crucially, unknown to the traveller, up to  $k$  edges of the graph are blocked and the traveller only discovers blocked edges online at one of their respective endpoints. The currently best known upper bound for  $k$ -CCTP is  $O(\sqrt{k})$  which was shown in [Huang and Liao, ISAAC '12]. We improve this polynomial bound to a logarithmic one by presenting a deterministic  $O(\log k)$ -competitive algorithm that runs in polynomial time. Further, we demonstrate the tightness of our analysis by giving a lower bound instance for our algorithm.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Online Algorithm, Canadian Traveller Problem, Travelling Salesperson Problem, Graph Exploration

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.53

**Funding** This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

**Acknowledgements** We would like to thank Evripidis Bampis and Bruno Escoffier for providing useful comments on the manuscript.

## 1 Introduction

The Canadian Traveller Problem (CTP) was introduced in 1991 by Papadimitriou and Yannakakis [26] as an extension of the Shortest Path Problem and has applications in online route planning in road networks. The goal of the problem is to find a shortest path between a source and a destination in an unreliable graph, in which some edges may become unavailable. This can only be observed in an online manner, i.e., when reaching one of the endpoints of such an edge. More specifically, consider a connected, undirected graph  $G = (V, E)$  with a source node  $s \in V$ , a destination node  $t \in V$  and a non-negative cost function  $c: E \rightarrow \mathbb{R}^+$  representing the cost to traverse each edge. A traveller seeks to find a path with minimum cost from  $s$  to  $t$ . However, one or more edges might be blocked, and thus cannot be traversed. The traveller only learns that an edge is blocked when reaching one of its endpoints. When the number of blocked edges is bounded by  $k$ , the variant is called  $k$ -Canadian Traveller Problem or  $k$ -CTP [6].

This work studies a generalization of CTP, defined in [22], which is called the Covering Canadian Traveller Problem (CCTP). In CCTP, one attempts to develop an efficient tour for a traveller that visits all vertices in a graph and returns to the origin (source) under the same uncertainty as that of CTP. When the number of blocked edges is bounded by  $k$ , the problem is called  $k$ -CCTP, analogous to the  $k$ -CTP variant of CTP.



© Niklas Hahn and Michalis Xeferis;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 53; pp. 53:1–53:12

Leibniz International Proceedings in Informatics



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

We make two assumptions on the underlying graph model, similar to [22]. First, we assume that the graph remains connected even if all blocked edges are removed. Second, the state of an edge, i.e., whether it is blocked or not, does not change after the traveller learns about it. The problem has practical uses in dynamic routing systems that prioritize efficient travel routes and aim to avoid traffic congestion. Since Huang and Liao introduced CCTP [22], there has been a notable amount of work on similar problems in the literature [1, 30, 36, 37]. For example, Zhang et al. [35] studied the Steiner Travelling Salesperson Problem in which the salesperson instantly learns about new blocked edges. Shiri et al. [29] focused on how to allocate and route search-and-rescue teams to areas with trapped victims, considering the uncertainty about road conditions which may delay the operations.

The motivation behind CCTP stems from other similar optimization problems, such as dynamic TSP and online TSP. Dynamic TSP has been studied for various different types of dynamic change, such as the addition or removal of locations, and changing pairwise distances between locations in the underlying space [21, 32]. On the other hand, Ausiello et al. [4] introduced the online TSP in which the input arrives over time, i.e., during the travel new requests (locations) appear that have to be visited by the algorithm. The problem has many practical applications, e.g., in logistics and robotics [2, 27]. Since its introduction, a series of papers has been published on the subject [3, 10, 19].

As is usual in the literature on online problems, we measure the performance of our algorithm by its competitive ratio [11]. This means that its performance is compared to the performance of an algorithm for the corresponding offline problem. In our setting, this would be an algorithm which knows the complete graph structure, including all blocked edges.

### Our Contribution

In this paper, we focus on  $k$ -CCTP. Currently, the best known deterministic algorithm for  $k$ -CCTP is the CYCLIC ROUTING algorithm by Huang and Liao [22] with competitive ratio  $O(\sqrt{k})$ . We improve this bound to  $O(\log k)$  by making a connection with the Online Graph Exploration problem. In the Online Graph Exploration problem, a searcher starts from a source vertex and aims to visit all vertices of an unknown but fixed graph. Upon reaching a new vertex, the server learns all incident edges and their costs. The reduction we give allows us to get a polynomial time algorithm for  $k$ -CCTP using an algorithm for the Exploration problem. Finally, we show that our analysis of the  $O(\log k)$ -competitive algorithm is tight.

## 2 Related Work

### Online Graph Exploration Problem

In the Online Graph Exploration problem, defined in [20], an agent has to explore an unknown graph by starting at a given vertex, visiting all other vertices, and returning to the starting one. The agent can only move along the edges of the graph and has to pay a cost for each traversed edge.

A simple and fast algorithm that solves the problem is the Nearest Neighbor (NN) algorithm. The algorithm selects an unexplored vertex that is cheapest to reach from the current one and visits it, repeating this process until all vertices are visited. This algorithm has been shown to have a competitive ratio of  $\Theta(\log n)$  for arbitrary graphs in the Online Graph Exploration Problem [28], which is a tight bound even on planar unit-weight graphs [15, 18]. Note that although the analysis in [28] deals with the offline problem, the nearest neighbor can always be identified even in the online scenario and the same analysis

applies. The second algorithm that achieves the  $\Theta(\log n)$  bound, which is the best known upper bound for arbitrary graphs, is the hierarchical Depth First Search algorithm (HDFS) in [23].

On the other hand, obtaining constant-competitive tours is known only for special cases of graphs, such as graphs with  $k$  distinct weights, graphs with bounded genus, cycles, tadpole graphs and cactus graphs [12, 15, 20, 23, 24]. Conversely, the best known lower bound on the competitive ratio of an online algorithm is just  $10/3$  [9], and despite efforts, it is still unclear whether there exists an  $o(\log n)$  or even  $O(1)$ -competitive exploration algorithm for general graphs.

### Canadian Traveller Problem

CTP has been proven to be PSPACE-complete [26]. For the  $k$ -CTP variant, Bar-Noy and Schieber proposed a polynomial time algorithm that minimizes the maximum travel length [6]. Westphal developed a simple deterministic online algorithm for  $k$ -CTP that is  $(2k + 1)$ -competitive and proved that no deterministic online algorithm can have a (strictly) better competitive ratio [33]. Furthermore, he showed a lower bound of  $k + 1$  for any randomized algorithm, even if all  $s - t$  paths are node disjoint. Xu et al. [34] proposed a deterministic algorithm that is also  $(2k + 1)$ -competitive for  $k$ -CTP and proved that a natural greedy strategy based on the available blockage information is exponential in  $k$ . On graphs where all  $s - t$  paths are node-disjoint, a  $(k + 1)$ -competitive randomized online algorithm is known [7, 31]. Demaine et al. [14] proposed a polynomial time randomized algorithm that improves the deterministic lower bound of  $2k + 1$  by an  $o(1)$  factor for arbitrary graphs. They also showed that the competitive ratio is even better if the randomized algorithm runs in pseudo-polynomial time. Recently, Bergé et al. [8] proved that the competitive ratio of any randomized algorithm using a specific set of strategies called memoryless cannot be better than  $2k + O(1)$ . Over the last few years, various other variants of CTP have been investigated [5, 17, 25].

### Covering Canadian Traveller Problem

The best known algorithm for  $k$ -CCTP is the one proposed in [22] with a competitive ratio of  $O(\sqrt{k})$ . The algorithm, called CYCLIC ROUTING, decomposes the entire route into several rounds. In each round, the traveller attempts to visit as many vertices as possible in the graph following the visiting order (or the reverse order) of the tour derived by Christofides' algorithm [13].

## 3 Preliminaries

In this section, we give some basic definitions. We start by giving a formal definition of the problem we study, before defining the Online Graph Exploration Problem, the performance measure and restating Christofides' algorithm for completeness. In what follows, we will denote by  $G = (V, E)$  a weighted, undirected graph. We will interchangeably use the notion of “cost” and “length” for the weight of an edge. For example, a shortest tour is a tour of minimum cost.

### Definition of $k$ -CCTP

The formal definition of CCTP is as follows. Given a complete metric graph  $G = (V, E)$  with a source vertex  $s \in V$ , a traveller aims, beginning from  $s$ , to visit every other vertex in  $V$  at least once and return to  $s$  with as little cost as possible. However, the traveller

discovers online that some edges are blocked once reaching one of their endpoints. Moreover, as mentioned earlier, two assumptions are made. First, the blocked edges cannot isolate vertices of  $G$ , i.e.,  $G$  remains connected, and second, edges remain in their state (i.e., whether they are blocked or not) forever. In this paper, we consider its variant  $k$ -CCTP where the number of blocked edges is bounded by  $k$ .

### Definition of the Online Graph Exploration Problem

The problem can be formalized as follows. Let  $G = (V, E)$  be a weighted, undirected graph with  $|V| = n$  vertices. The agent starts at a vertex  $s \in V$  and has to visit every vertex in the graph and return to  $s$ . Note that the agent can visit a vertex more than once. At each step, the agent is located at a vertex  $u$  and can choose to move to any of the neighboring vertices of  $u$ . The agent incurs a cost equal to the cost of the edge traversed. Upon arriving at a vertex  $v$ , the agent learns all the edges incident to  $v$  and their costs.

### Competitive Ratio

A deterministic online algorithm  $ALG$  for  $k$ -CCTP is  $c$ -competitive if the total cost  $|ALG(\sigma)|$  accrued by  $ALG$  for input  $\sigma$  is at most  $c \cdot |OPT(\sigma)|$ . Here,  $|OPT(\sigma)|$  is the total cost of an optimal tour for  $\sigma$  which is computed by an offline algorithm that already knows all blocked edges.

### Christophides' algorithm

We also remind the reader how Christophides' algorithm works. Christophides' algorithm on a complete metric graph  $G$  can be described as follows [16]:

1. Create a minimum spanning tree  $T$  of  $G$ .
2. Find a minimum-weight perfect matching  $M$  in the subgraph of  $G$  that is induced by the vertices with odd degrees in  $T$ .
3. Combine the edges of  $M$  and  $T$  to form a connected multigraph  $H$ .
4. Form a Eulerian cycle in  $H$ .
5. Make the circuit found in the previous step into a Hamiltonian cycle by skipping repeated vertices.

## 4 Solving $k$ -CCTP via Graph Exploration

In this section, we present the results of our work. First, we show a connection between CCTP and the Online Graph Exploration problem (Theorem 1). This is the crucial step to improve the upper bound of  $O(\sqrt{k})$ .

The idea behind our reduction is that CCTP can be solved by an algorithm that solves the Online Graph Exploration Problem. This is possible since at every step, the traveller locally learns the real edges in both problems. The challenge here is that the algorithms for the Online Graph Exploration for arbitrary graphs have competitive ratios depending on the number of vertices  $n$ .

So, how can we reduce the size of the graph in which we run an algorithm for Graph Exploration to something of size  $O(k)$ ? First, we try to follow an approximately optimal TSP tour, skipping vertices when edges are discovered to be blocked. Similar to the idea in CYCLIC ROUTING of [22], we use a function SHORTCUT to achieve this. After that, we return to the starting vertex. This way, we visit at least  $n - k$  vertices of  $G$ . Since they do not have to be visited again, we can then use the information gathered to reduce the number of vertices in the graph on which we will run the algorithm for Graph Exploration to  $O(k)$ . Formally, we have the following theorem.

► **Theorem 1.** *If there exists an  $f(k)$ -competitive algorithm for the Online Graph Exploration problem on graphs with at most  $k + 1$  nodes and an  $\alpha$ -approximation algorithm for metric TSP, then there exists an  $(f(k) + 2\alpha)$ -competitive algorithm for  $k$ -CCTP.*

**Proof.** Suppose that we have ALGOEXPLORATION, an  $f(k)$ -competitive algorithm for the Online Graph Exploration problem on arbitrary graphs with at most  $k + 1$  nodes, and ALGOTSP, an  $\alpha$ -approximation algorithm for metric TSP. Then, we will prove that the algorithm COMPRESSANDEXPLORE that uses these algorithms as subroutines has a competitive ratio of at most  $f(k) + 2\alpha$  for  $k$ -CCTP (for pseudocode, see Algorithm 1).

First, the algorithm runs ALGOTSP on the input graph  $G$  to compute a TSP tour  $P$ . For simplicity, we relabel the vertices with respect to the tour, i.e., we assume that the tour  $P$  has the order  $s = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . If an edge  $\{v_i, v_j\}$  is blocked, the traveller tries to go to the next vertex in the order determined by  $P$ , i.e.,  $v_{j+1}$ , or  $v_1$  for  $j = n$  (for pseudocode of this subroutine, see Function SHORTCUT on page 8). This procedure is possible since the original graph is complete. By the triangle inequality, the cost of the tour is upper bounded by the cost of tour  $P$ . If the traveller reaches vertex  $s$ , then SHORTCUT terminates. If  $s$  is not reachable directly because of a blocked edge, the traveller returns to  $s$  by retracing their steps. Since  $\text{cost}(P)$  is an  $\alpha$ -approximation for metric TSP, we have that  $\text{cost}(\text{SHORTCUT}) \leq 2 \cdot \text{cost}(P) \leq 2\alpha \cdot |\text{OPT}|$ , where  $\text{OPT}$  is an optimal offline TSP tour on graph  $G$ .

The traveller learns about all blocked edges which are adjacent to the vertices that are visited during SHORTCUT. In the procedure, all edges that are discovered to be blocked are collected in the set  $E_b$ . Thus, the traveller knows the whole graph (with all blocks) except for the induced (complete) subgraph formed by the unvisited vertices  $U$ . Let  $\kappa$  be the number of vertices which remain unvisited by  $ALG$  after SHORTCUT, i.e., the size of the set  $U$ . Then, the traveller has discovered at least  $\kappa$  blocked edges, i.e.,  $|E_b| \geq \kappa$ .

Next, the traveller, being at  $s$ , has to visit the vertices in  $U$ . Since the true edges of the graph except for those of the induced subgraph formed by vertices in  $U$  are known, it suffices to consider only the vertices in the set  $U_s = U \cup \{s\}$ . While the vertices in  $V \setminus U_s$  themselves are not required, a shortest path between two vertices  $x, y \in U_s$  might include vertices from the set  $V \setminus U_s$  as intermediate nodes. This can occur when currently unknown edges between unvisited nodes are blocked. More specifically, the algorithm runs the function COMPRESS (for pseudocode, see Function COMPRESS on page 9). For every pair of vertices  $x, y \in U_s$ , the function creates a new edge  $P_{x,y}$  representing a shortest path between  $x$  and  $y$  such that the path consists only of edges that are known not to be blocked, i.e., edges in which at least one node has already been visited before – if such a shortest path exists. Note that this phase of the algorithm does not incur any cost in terms of competitive ratio. Thus, the procedure creates a multigraph  $G'$  which consists of vertex set  $U_s$ , the initial edges that connect these vertices and the “shortest-path” edges as described above. To better explain the steps of the algorithm we present an example of an execution of algorithm COMPRESSANDEXPLORE below (see Example 1).

Finally, the algorithm runs ALGOEXPLORATION on  $G'$  and visits the remaining vertices.<sup>1</sup> Every time the traveller visits a vertex, they learn all incident edges. This includes the newly added “shortest-path” edges, of which we know that they are feasible. If the traveller uses

<sup>1</sup> ALGOEXPLORATION solves the Exploration problem on arbitrary graphs, but  $G'$  is a multigraph with at most two edges per pair of vertices. However, this does not cause a problem, since the algorithm can always select a shortest edge out of the two and the optimal solution can be computed while keeping only one edge per pair.

such a “shortest-path” edge, then in the final computed tour in the original graph we expand it, meaning that we use the real path that corresponds to this edge. The cost of an optimal TSP tour  $OPT_{G'}$  on multigraph  $G'$  is at most the cost of an optimal TSP tour  $OPT_G(U_s)$  that only has to visit the vertex set  $U_s$ , i.e., the vertices that are also in  $G'$ , but inside the input graph  $G$ . To see that this holds, consider an optimal tour  $OPT_G(U_s)$ . Assume that it visits the vertices in  $U_s$  in the order  $s = x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_{|U_s|} \rightarrow x_1$ . Between any vertices  $x_i, x_{i+1} \in U_s$  for  $i \in \{1, \dots, |U_s| - 1\}$  (or  $x_{|U_s|}, x_1$ ) that are visited one after the other,  $OPT_G(U_s)$  uses a shortest path. Each of these shortest paths starts in  $U_s$ . If it then uses an edge to another vertex in  $U_s$ , this edge will also be in  $G'$  as each direct edge between two vertices of  $U_s$  will either be blocked in both  $G$  and  $G'$  or not be blocked in both. Hence, we can assume that an edge  $\{u, v\}$  from  $u \in U_s$  to a vertex  $v \notin U_s$  is taken. This is an already discovered edge, as  $v$  has already been visited during SHORTCUT. Eventually, the path will re-enter into the set  $U_s$  by using another already discovered edge  $\{v', u'\}$  for some  $v' \notin U_s$  and  $u' \in U_s$ . In between leaving and re-entering, all edges that were taken are also already discovered and this partial path has exactly the same length as the shortest-path edge  $P_{u,u'}$  between  $u, u' \in U_s$ .<sup>2</sup> Continuing this argument, eventually the target vertex in  $U_s$  is reached. All intermediate partial paths are inside  $G'$ , either since they are regular edges that also exist in  $G$ , or since they have been added as shortest-path edges during SHORTCUT.

The multigraph  $G'$  has  $\kappa + 1$  vertices and at least  $\kappa$  blocks have already been discovered. The number of blocked edges is at most  $k$ , and thus there are at most  $k + 1$  vertices in  $G'$ . So, from the hypothesis the cost incurred by ALGOEXPLORATION on  $G'$  is at most  $f(k) \cdot |OPT_{G'}|$ . Since an optimal solution for visiting a subset of vertices  $OPT_G(U_s)$  has cost at most  $|OPT|$ , we get the following

$$\text{cost}(\text{ALGOEXPLORATION}) \leq f(k) \cdot |OPT_{G'}| \leq f(k) \cdot |OPT_G(U_s)| \leq f(k) \cdot |OPT| .$$

Overall, the algorithm has a total cost for the traveller of

$$\begin{aligned} \text{cost}(\text{COMPRESSANDEXPLORE}) &= \text{cost}(\text{SHORTCUT}) + \text{cost}(\text{ALGOEXPLORATION}) \\ &\leq (f(k) + 2\alpha) \cdot |OPT| . \end{aligned}$$

Consequently, COMPRESSANDEXPLORE is an  $(f(k) + 2\alpha)$ -competitive algorithm for  $k$ -CCTP. Note that the knowledge of  $k$  does not affect the performance of the algorithm. ◀

► **Remark.** In the proof, we allow  $k \geq n - 1$  as long as the resulting graph remains connected. The analysis of the competitive ratio of  $O(\sqrt{k})$  in [22] requires  $k < n - 1$ .

► **Example 1.** Fig. 1 shows an example of COMPRESSANDEXPLORE. The traveller begins at vertex  $s = v_1$  and moves in a counterclockwise direction. The given TSP tour by ALGOTSP here is  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_{16} \rightarrow v_1$ . The solid lines represent the tour that the traveller follows during SHORTCUT due to the discovered blocked edges (red dashed lines). The traveller follows the shortcut path  $v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_5 \rightarrow v_9 \rightarrow v_{10} \rightarrow v_{11} \rightarrow v_{14} \rightarrow v_{16}$  and after visiting vertex  $v_{16}$ , they return back to  $s$  following the same path backwards. Next, the algorithm runs COMPRESS and gets  $G'$ . Multigraph  $G'$  contains  $s$ , the remaining unvisited vertices and at most two edges between each pair of these vertices. Between  $v_i$  and  $v_j$  there is the edge  $\{v_i, v_j\}$  (which may be blocked) and possibly the “shortest-path” edge  $P_{i,j}$ . The cost of  $P_{i,j}$

<sup>2</sup> There might be several shortest paths with the same length, which is why the shortest path chosen for  $P_{u,u'}$  and the described shortest path might differ. Still, their lengths are equal by definition.

■ **Algorithm 1** COMPRESSANDEXPLORE(ALGOTSP, ALGOEXPLORATION).

---

**Input** : A complete metric graph  $G = (V, E)$  with  $n$  vertices; a starting vertex  $s \in V$ ;

**Output** : A tour that visits every vertex in  $V$ ;

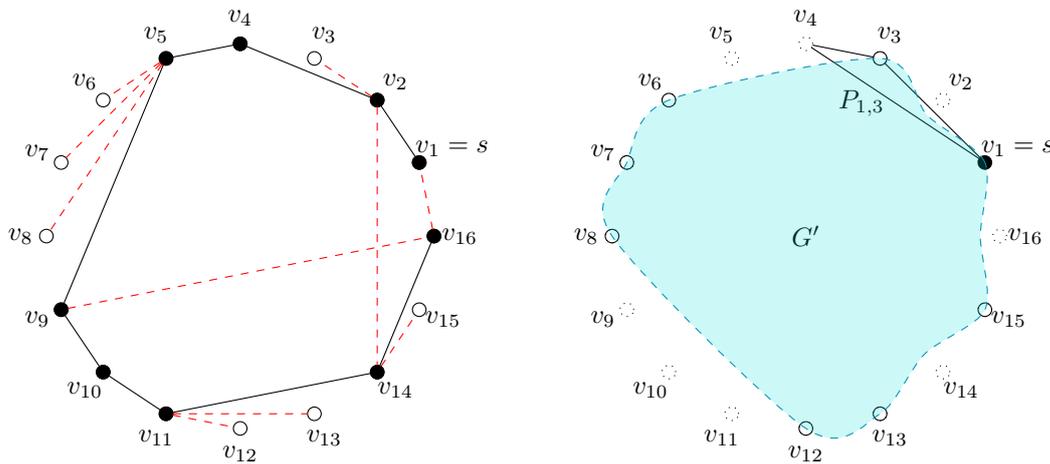
**Parameter**: ALGOTSP( $G_1$ ): An algorithm that returns a TSP tour on a metric graph  $G_1$ ; The tour has the form  $s = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ ;  
 ALGOEXPLORATION( $G_2$ ): An algorithm that solves the Online Graph Exploration problem on an arbitrary graph  $G_2$  and returns a tour;

- 1  $P \leftarrow \text{ALGOTSP}(G)$ ;
- 2  $G^*, U, P_1 \leftarrow \text{SHORTCUT}(G, P)$ ;
- 3  $G' \leftarrow \text{COMPRESS}(G^*, U, G)$ ;
- 4  $P_2 \leftarrow \text{ALGOEXPLORATION}(G')$ ;
- 5  $P' \leftarrow (P_1 \rightarrow P_2)$ ;

/\* Concatenate  $P_1$  and  $P_2$ , i.e., visit the vertices according to  $P_1$ , then according to  $P_2$ . \*/

6 **return**  $P'$ ;

---



■ **Figure 1** An example of algorithm COMPRESSANDEXPLORE.

is the cost of the shortest path from  $v_i$  to  $v_j$  in which each edge has at least one endpoint outside of  $G'$ . In the example, a possible case for  $i = 1$  and  $j = 3$  is shown on the right with  $P_{1,3}$  being the path  $v_1 \rightarrow v_4 \rightarrow v_3$ .

Finally, the algorithm runs ALGOEXPLORATION on  $G'$ . The traveller visits all remaining vertices, returns to  $s$  and the algorithm terminates.

Now, we can use COMPRESSANDEXPLORE with Christophides' algorithm for metric TSP and the Nearest Neighbor for the Online Graph Exploration problem. Since this algorithm uses Christophides' algorithm and then Nearest Neighbor, we refer to it by CNN.

► **Corollary 2.** CNN has a competitive ratio of  $O(\log k)$  for  $k$ -CCTP.

**Proof.** Christophides' algorithm gives a  $3/2$ -approximation for metric TSP. On the other hand, NN yields a competitive ratio of  $O(\log n)$  for the Graph Exploration problem in an arbitrary graph, where  $n$  is the number of vertices in the graph. Thus, from Theorem 1 we get that CNN has a competitive ratio of  $3 + O(\log(k + 1)) = O(\log k)$ . ◀

---

```

1 Function SHORTCUT( $G, P$ ):
   | /*  $G$  is the input graph and  $P$  a TSP tour. */
   | /*  $P$  has the form  $s = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ . */
2   |  $i \leftarrow 1; j \leftarrow 2;$ 
3   |  $U_s \leftarrow \{s\}; E_b \leftarrow \emptyset; P' \leftarrow \{s\};$ 
   | /* Path  $P'$  which the traveller follows is built. */
4   | while  $j \leq n$  do
5   |   | Add all newly discovered blocked edges  $\{v_i, x\}$ , with  $x \in V \setminus \{v_i\}$ , to  $E_b$ ;
6   |   | if  $\{v_i, v_j\}$  is not blocked then
7   |   |   |  $P' \leftarrow (P' \rightarrow v_j);$  // Append  $v_j$  to  $P'$ 
8   |   |   |  $i \leftarrow j;$ 
9   |   |   | else
10  |   |   |   |  $U_s \leftarrow U_s \cup \{v_j\};$ 
11  |   |   |   | end
12  |   |   |  $j \leftarrow j + 1;$ 
13  |   | end
14  |   | if  $\{v_i, v_1\}$  is blocked then
15  |   |   | Return to  $s$  following  $P'$  backwards;
16  |   |   |  $P' \leftarrow$  Concatenate the path  $P'$  and the reverse of  $P'$  to return to  $s$ ;
17  |   |   | else
18  |   |   |   |  $P' \leftarrow (P' \rightarrow v_1);$ 
19  |   |   |   | end
20  |   |  $G^* \leftarrow (V, E \setminus E_b);$ 
21  |   | return  $G^*, U_s, P'$ ;
22 end Function

```

---

To demonstrate that the above analysis is tight, the following theorem presents a family of instances that achieves a competitive ratio of  $\Omega(\log k)$  and therefore proves the analysis to be tight.

► **Theorem 3.** *There exists a family of instances for which CNN has a competitive ratio of the  $\Omega(\log k)$ .*

**Proof.** We will use the graph presented in [18] to lower bound the competitive ratio of the Nearest Neighbor algorithm. For an integer  $p \geq 1$ , the graph  $G_p = (V_p, E_p)$  consists of a chain of  $2^p - 1$  triangles.  $G_p$  has  $2^p$  vertices in its lower level, and  $2^p - 1$  vertices in its upper level. The left-most vertex in the lower level is denoted by  $l_p$ , the right-most by  $r_p$  and the central vertex in the upper level is denoted by  $m_p$ . All edges in  $G_p$  have an equal cost of 1.

For our instance, we slightly modify the graph by adding another vertex  $u$  to the left of  $l_p$  with an edge  $\{u, l_p\}$  of cost 1. We also set  $s = l_p$  as the starting vertex. Since the input for  $k$ -CCTP is a complete graph, we also need to add some more edges. All edges from  $u$  to the other vertices have a cost of 1, and all other new edges have a cost of 2. All these edges will be blocked edges. We call this new graph  $G_p^+$ .

The resulting graph has  $k = \Theta(n^2)$  blocked edges and clearly satisfies the triangle inequality. We illustrate the non-blocked part of  $G_p^+$  for  $p = 3$  in Fig. 2.

In the first step of Christophides' algorithm, a minimal spanning tree is constructed. One possible MST  $T$  is a path from  $u$  to  $r_p$ . The nodes with uneven degree in  $T$  are the nodes  $u$  and  $r_p$ , so for the matching, the edge between  $u$  and  $r_p$  is added. This results in a simple cycle of all nodes. The MST and the matching edge are illustrated in Fig. 3.

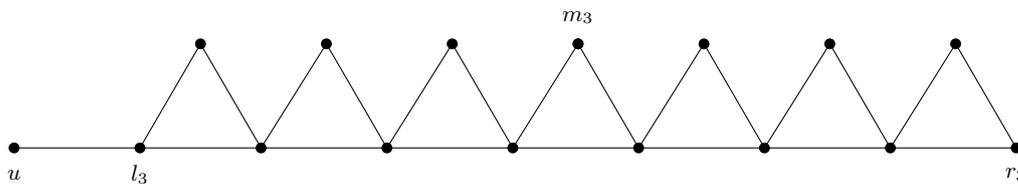
---

```

1 Function COMPRESS( $G^*, U_s, G$ ):
   /*  $G^*$  is the graph without the discovered blocked edges and  $U_s$  the
   set of the remaining unvisited vertices in the graph together
   with the starting vertex  $s$ . */
2  $E' \leftarrow \{\{x, y\} \in E \mid x, y \in U_s\}$ ;
   /*  $E'$  is the subset of edges with unknown state, i.e., of  $\{x, y\}$ 
   with  $x, y \in U_s$ . */
3  $G' \leftarrow (U_s, E')$ ;
4  $H \leftarrow (V, E \setminus E')$ ;
   /*  $H$  includes all edges with a known state, since in every edge at
   least one vertex has already been visited. */
5 Let  $U_s = \{v'_1, v'_2, \dots, v'_{|U_s|}\}$ ;
6 for  $i \leftarrow 1$  to  $|U_s|$  do
7   for  $j \leftarrow i + 1$  to  $|U_s|$  do
8     Find a shortest path  $P_{i,j}$  from  $v'_i$  to  $v'_j$  in  $H$ ;
9      $c_{i,j} \leftarrow$  total cost of  $P_{i,j}$ ;
10    Add an edge  $\{v'_i, v'_j\}$  with cost  $c_{i,j}$  to  $G'$ ;
11  end
12 end
13 return  $G'$ ;
14 end Function

```

---



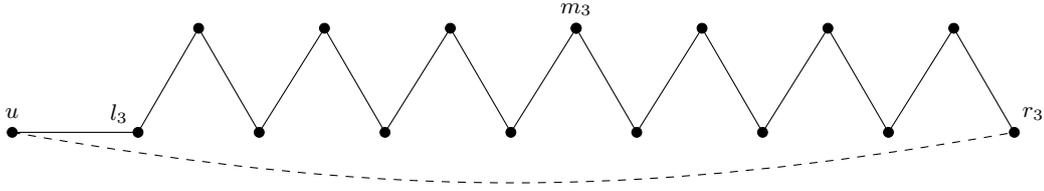
■ **Figure 2** Graph  $G_p^+$  (for  $p = 3$ ).  $G_p^+$  is used to show tightness of the  $O(\log k)$ -competitive ratio.

The TSP-tour can then be chosen to be  $s = l_3 \rightarrow u \rightarrow r_3 \rightarrow \dots \rightarrow l_3$ . This means that in SHORTCUT, only node  $u$  would be visited besides  $l_p$  as the direct edges from  $u$  to any other node (besides  $l_p$ ) are blocked. At the end of SHORTCUT, the traveller returns to  $l_p$ .

After SHORTCUT, the remaining graph would thus be the original graph  $G_p$  from [18]. We use the following lemma to prove that there exists a TSP-tour in  $G_p$  which starts (and ends) in  $l_p$  which is found by NN that has a length of  $(p + 4) \cdot 2^{p-1} - 2$ . We will prove the lemma below.

► **Lemma 4** (Based on [18, Lemma 1]). *There exists a NN-based TSP tour on  $G_p$  which starts in  $l_p$  and visits  $m_p$  as final vertex before returning back to  $l_p$ . The tour has length  $(p + 4) \cdot 2^{p-1} - 2$ .*

Using this result, the total cost of the described TSP-tour is  $(p + 4)2^{p-1}$ , whereas an optimal TSP-tour has cost  $2 + 3(2^p - 1)$ , namely visiting  $u$  and optimally visiting  $G_p$  by going in a zig-zag motion from left to right (as shown in the MST in Fig. 3) and returning using the lower edges of the triangle, thereby using each edge of the triangles exactly once.



■ **Figure 3** The MST and the matching edge (dashed line) in  $G_3^+$ .

This gives us a ratio of

$$\frac{(p+4) \cdot 2^{p-1}}{2+3 \cdot (2^p-1)} = \frac{(p+4) \cdot 2^{p-1}}{6 \cdot 2^{p-1}-1} \geq \frac{p+4}{6} = \Omega(p) = \Omega(\log n) . \quad \blacktriangleleft$$

**Proof of Lemma 4.** We split the tour into two parts. In the first part, all vertices are visited, and in the second part, the traveller returns to  $l_p$ .

The second part has length  $1 + 2^{p-1} - 1 = 2^{p-1}$ . This is true because the whole graph has been discovered and the traveller can take the shortest path from  $m_p$  to  $l_p$ , which is going down to the left point of the middle triangle and then traversing the  $2^{p-1} - 1$  many triangles on the left side to reach  $l_p$ .

Hence, to show the Lemma, it remains to show that there exists a NN-route to visit all vertices in  $G_p$  which has length  $(p+3) \cdot 2^{p-1} - 2 = (p+4) \cdot 2^{p-1} - 2 - 2^{p-1}$ , starting at  $l_p$  and ending at  $m_p$ . We prove this by induction. For  $p=1$ ,  $G_p$  consists of a single triangle, and the route  $l_1 \rightarrow r_1 \rightarrow m_1$  has length  $2 = (1+3) \cdot 2^{1-1} - 2$ .

For the inductive step, we observe that  $G_p$  can be constructed from two copies of  $G_{p-1}$  and an additional vertex  $m_p$  (and three additional edges). Let  $G_{p-1}^{(l)}$  be the left copy and  $G_{p-1}^{(r)}$  be the right copy. Then, the new edges are  $\{r_{p-1}^{(l)}, l_{p-1}^{(r)}\}$ ,  $\{r_{p-1}^{(l)}, m_p\}$  and  $\{l_{p-1}^{(r)}, m_p\}$ . This is illustrated in Fig. 4. By the induction hypothesis, there exists an NN-route in  $G_{p-1}^{(l)}$  starting in  $l_p = l_{p-1}^{(l)}$  and ending in  $m_{p-1}^{(l)}$  with length  $(p-1+3) \cdot 2^{p-1-1} - 2$ . The two nearest unvisited neighbors to  $m_{p-1}^{(l)}$  are  $m_p$  and  $l_{p-1}^{(r)}$  with equal distance  $2^{p-2} + 1$ . By going to  $l_{p-1}^{(r)}$ , the sub-route from  $l_{p-1}^{(r)}$  to  $m_{p-1}^{(r)}$  of length  $(p+2) \cdot 2^{p-2} - 2$  can then be found by NN. Note that throughout this route,  $m_p$  will never be closer to the current vertex than any other unvisited vertex in the current sub-route and thus will not be visited before  $m_{p-1}^{(r)}$ . Finally,  $m_p$  needs to be visited, which requires an additional cost of  $2^{p-2} + 1$ . Overall, there exists an NN-route from  $l_p$  to  $m_p$  with length

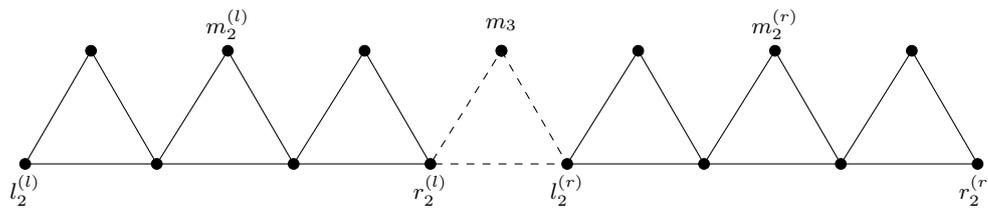
$$2 \cdot ((p+2) \cdot 2^{p-2} - 2) + 2 \cdot (2^{p-2} + 1) = (p+2) \cdot 2^{p-1} - 4 + 2^{p-1} + 2 = (p+3) \cdot 2^{p-1} - 2 .$$

This concludes the proof. ◀

Finally, we remark that CNN takes polynomial time. The procedures SHORTCUT and COMPRESS run in polynomial time as the required shortest paths can be computed in polynomial time. Since Christophides' algorithm and Nearest Neighbor also have polynomial time complexity, so does CNN.

## 5 Concluding Remarks

In this work, we considered the Covering Canadian Traveller Problem with up to  $k$  blocked edges. We improved the upper bound to  $O(\log k)$  by drawing an interesting connection to the Online Graph Exploration problem. Further, we showed the tightness of our analysis.



■ **Figure 4** Graph  $G_p$  (for  $p = 3$ ) constructed from two copies of  $G_{p-1}$ , with the joining edges denoted by the dashed lines.

Our reduction implies immediate consequences of future work on the respective other problem. For one, it allows an improvement of the lower bound on the Graph Exploration problem using a general lower bound on  $k$ -CCTP. Currently, the best known bound for the Graph Exploration problem is  $10/3$ . Tightening this gap would be a very interesting result. Second, an improved algorithm for the Graph Exploration problem immediately gives rise to a better algorithm and upper bound on  $k$ -CCTP.

Nevertheless, already an improved algorithm for  $k$ -CCTP or a lower bound on the Graph Exploration problem would be of independent interest without exploiting our reduction and thus provides another challenging direction of future research.

---

## References

- 1 Vahid Akbari and Davood Shiri. Weighted online minimum latency problem with edge uncertainty. *Europ. J. Oper. Res.*, 295(1):51–65, 2021.
- 2 Norbert Ascheuer, Martin Grötschel, Sven O. Krumke, and Jörg Rambau. Combinatorial online optimization. In *Operations Research Proceedings 1998*, pages 21–37, 1999.
- 3 Giorgio Ausiello, Marc Demange, Luigi Laura, and Vangelis Paschos. Algorithms for the on-line quota traveling salesman problem. *Inf. Process. Lett.*, 92(2):89–94, 2004.
- 4 Giorgio Ausiello, Esteban Feuerstein, Stefan Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Synthetic Metals*, 1999.
- 5 Evripidis Bampis, Bruno Escoffier, and Michalis Xefteris. Canadian traveller problem with predictions. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms*, pages 116–133, Cham, 2022. Springer International Publishing.
- 6 Amotz Bar-Noy and Baruch Schieber. The canadian traveller problem. In *Proc. 2nd Symp. Disc. Algorithms (SODA)*, pages 261–270, 1991.
- 7 Marco Bender and Stephan Westphal. An optimal randomized online algorithm for the  $k$ -canadian traveller problem on node-disjoint paths. *J. Comb. Opt.*, 30, July 2013.
- 8 Pierre Bergé, Julien Hemery, Arpad Rimmel, and Joanna Tomasik. On the competitiveness of memoryless strategies for the  $k$ -Canadian Traveller Problem. In *Int. Conf. Comb. Opt. Appl. (COCOA)*, December 2018.
- 9 Alexander Birk, Yann Disser, Alexander V. Hopp, and Christina Karousatou. An improved lower bound for competitive graph exploration. *Theor. Comp. Sci.*, 868:65–86, 2021.
- 10 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online TSP on the line. *ACM Trans. Algorithms*, 17(1):3:1–3:58, 2021.
- 11 Allan Borodin and Ran El-Yaniv. Online computation and competitive analysis. In *Cambridge University Press*, 1998.
- 12 Sebastian Brandt, Klaus-Tycho Foerster, Jonathan Maurer, and Roger Wattenhofer. Online graph exploration on a restricted graph class: Optimal solutions for tadpole graphs. *Theor. Comp. Sci.*, 839:176–185, 2020.

- 13 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3, 1976.
- 14 Erik D. Demaine, Yamming Huang, Chung-Shou Liao, and Kunihiro Sadakane. Approximating the canadian traveller problem with online randomization. *Algorithmica*, 83(5):1524–1543, 2021.
- 15 Robin Fritsch. Online graph exploration on trees, unicyclic graphs and cactus graphs. *Inf. Process. Lett.*, 168:106096, 2020.
- 16 Michael T. Goodrich and Roberto Tamassia. *Algorithm design and applications*. Wiley, 2015.
- 17 Yamming Huang and Chung-Shou Liao. The canadian traveller problem revisited. In *Proc. 23rd Int. Symp. Algorithms and Comp. (ISAAC)*, pages 352–361, 2012.
- 18 Cor A.J. Hurkens and Gerhard Woeginger. On the nearest neighbor rule for the traveling salesman problem. *Oper. Res. Lett.*, 32(1):1–4, 2004.
- 19 Patrick Jaillet and Xin Lu. Online Traveling Salesman Problems with Flexibility. In *Models and Algorithms for Optimization in Logistics*, volume 9261 of *Dagstuhl Seminar Proceedings (DagSemProc)*, pages 1–4, 2009.
- 20 Bala Kalyanasundaram and Kirk R. Pruhs. Constructing competitive tours from local information. *Theor. Comp. Sci.*, 130(1):125–138, 1994.
- 21 Allan Larsen, Oli Madsen, and Marius Solomon. The a priori dynamic traveling salesman problem with time windows. *Transp. Sci.*, 38:459–472, November 2004.
- 22 Chung-Shou Liao and Yamming Huang. The covering canadian traveller problem. *Theor. Comp. Sci.*, 530:80–88, 2014.
- 23 Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theor. Comp. Sci.*, 463:62–72, 2012.
- 24 Shuichi Miyazaki, Naoyuki Morimoto, and Yasuo Okabe. The online graph exploration problem on restricted graphs. *IEICE Transactions*, 92-D:1620–1627, September 2009.
- 25 Evdokia Nikolova and David R. Karger. Route planning under uncertainty: The canadian traveller problem. In *Proc. 23rd Conf. Artif. Intell. (AAAI)*, pages 969–974, 2008.
- 26 Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.
- 27 Harilaos N. Psaraftis, Marius M. Solomon, Thomas L. Magnanti, and Tai-Up Kim. Routing and scheduling on a shoreline with release times. *Manag. Sci.*, 36(2):212–223, 1990.
- 28 Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis. *An analysis of several heuristics for the traveling salesman problem*, pages 45–69. Springer Netherlands, 2009.
- 29 Davood Shiri, Vahid Akbari, and F. Sibel Salman. Online routing and scheduling of search-and-rescue teams. *OR Spectrum*, 42:1–30, September 2020.
- 30 Davood Shiri and F. Sibel Salman. On the online multi-agent O–D  $k$ -Canadian Traveler Problem. *J. Comb. Opt.*, 34, August 2017.
- 31 Davood Shiri and F. Sibel Salman. On the randomized online strategies for the  $k$ -canadian traveler problem. *J. Comb. Opt.*, 38:254–267, 2019.
- 32 Alejandro Toriello, William Haskell, Michael Poremba, and Daniel Epstein. A dynamic traveling salesman problem with stochastic arc costs. *Oper. Res.*, 62, October 2014.
- 33 Stephan Westphal. A note on the  $k$ -canadian traveller problem. *Inf. Process. Lett.*, 106(3):87–89, 2008.
- 34 Yinfeng Xu, Maolin Hu, Bing Su, Binhai Zhu, and Zhijun Zhu. The canadian traveller problem and its competitive analysis. *J. Comb. Optim.*, 18(2):195–205, 2009.
- 35 Huili Zhang, Weitian Tong, Yinfeng Xu, and Guohui Lin. The steiner traveling salesman problem with online edge blockages. *Europ. J. Oper. Res.*, 243(1):30–40, 2015.
- 36 Huili Zhang, Weitian Tong, Yinfeng Xu, and Guohui Lin. The steiner traveling salesman problem with online advanced edge blockages. *Computers & Operations Research*, 70:26–38, 2016.
- 37 Yubai Zhang, Zhao Zhang, Zhaohui Liu, and Qirong Chen. An asymptotically tight online algorithm for  $m$ -steiner traveling salesman problem. *Inf. Process. Lett.*, 174:106177, 2022.

# On the Complexity of Computing Time Series Medians Under the Move-Split-Merge Metric

Jana Holznigenkemper ✉

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Christian Komusiewicz ✉ 

Institute of Computer Science, Friedrich-Schiller-Universität Jena, Germany

Nils Morawietz ✉

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Bernhard Seeger ✉

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

---

## Abstract

We initiate a study of the complexity of MSM-MEDIAN, the problem of computing a median of a set of  $k$  real-valued time series under the move-split-merge distance. This distance measure is based on three operations: moves, which may shift a data point in a time series; splits, which replace one data point in a time series by two consecutive data points of the same value; and merges, which replace two consecutive data points of equal value by a single data point of the same value. The cost of a move operation is the difference of the data point value before and after the operation, the cost of split and merge operations is defined via a given constant  $c$ .

Our main results are as follows. First, we show that MSM-MEDIAN is NP-hard and W[1]-hard with respect to  $k$  for time series with at most three distinct values. Under the Exponential Time Hypothesis (ETH) our reduction implies that a previous dynamic programming algorithm with running time  $|I|^{\mathcal{O}(k)}$  [Holznigenkemper et al., Data Min. Knowl. Discov. '23] is essentially optimal. Here,  $|I|$  denotes the total input size. Second, we show that MSM-MEDIAN can be solved in  $2^{\mathcal{O}(d/c)} \cdot |I|^{\mathcal{O}(1)}$  time where  $d$  is the total distance of the median to the input time series.

**2012 ACM Subject Classification** Mathematics of computing → Time series analysis

**Keywords and phrases** Parameterized Complexity, Median String, Time Series, ETH

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.54

**Funding** *Jana Holznigenkemper*: Partially supported by the LOEWE initiative (Hesse, Germany) within the emergenCITY center.

*Nils Morawietz*: Supported by the DFG, project OPERAH, KO 3669/5-1.

## 1 Introduction

Computing an exact or approximate consensus of a set of real-valued time series is a critical task in many applications like nearest-neighbor classification and clustering of time series [1, 8, 10]. The algorithmic task in this problem is to compute for a given set  $X$  of time series some time series  $x$  which has a small distance to the members of  $X$ . Naturally, the quality of the computed consensus time series for the above-mentioned applications and the difficulty of computing such a consensus depend highly on the underlying time series distance function.

The best results in terms of quality and robustness of the computed distances is achieved by elastic distance measures [11]. Informally, when computing an elastic distance measure for two time series, each time series may be stretched or compressed. This ensures that the best-fitting parts are aligned. Elastic distance measures allow for comparison of time series of different lengths, are translation invariant and robust to temporal misalignment [9]. The high quality of elastic measures however comes at a high running time cost [11].



© Jana Holznigenkemper, Christian Komusiewicz, Nils Morawietz, and Bernhard Seeger; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 54; pp. 54:1–54:15

Leibniz International Proceedings in Informatics



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

Consider for example the most popular elastic measure, the *dynamic time warping* (DTW) distance. The DTW distance of two time series of length  $n$  and  $m$  can be computed in  $\mathcal{O}(nm)$  time by a simple dynamic programming (DP) algorithm. Under the SETH, a fairly common complexity-theoretic assumption, this cannot be improved to  $\mathcal{O}(n^{2-\epsilon})$  time even when both time series have length  $n$  [3]. DTW-MEAN, the problem of computing a mean under the DTW distance measure, that is, a time series that minimizes the sum of squared DTW distances to the input time series, can be solved in  $\mathcal{O}(n^{2k+1}2^k k)$  time [2]. Here,  $n$  is the maximum time series length and  $k$  is the number of input time series. While this running time is polynomial for a constant number of strings, it grows rapidly with  $k$  making the algorithm impractical for  $k \geq 4$ . DTW-MEAN is NP-hard [4], so a polynomial-time algorithm cannot be expected. Moreover, DTW-MEAN is W[1]-hard with respect to  $k$  and thus an FPT algorithm for  $k$ , that is, an algorithm with running time  $f(k) \cdot n^{\mathcal{O}(1)}$  is presumably impossible [4]. Altogether, the complexity of distance and mean computation under the DTW measure are fairly well-understood by now.

DTW is by far not the only elastic distance measure. In this work, we study the *Move-Split-Merge* (MSM) distance [12]. Here, one time series is transformed into another using three types of operations. A *move* shifts one value in a time series, a *split* replaces one point of a time series by two subsequent points of the same value and a *merge* replaces two subsequent points of the same value by a single point of the same value. In contrast to DTW, the MSM distance fulfills the properties of a metric. Moreover, it achieves high accuracy in 1-NN classification tasks [9, 11]. This highly motivates an algorithmic study of distance computation and median finding problems. Initially, the distance computation for the MSM metric was found to be much slower than for DTW [11]. Recent progress showed that, on the practical side, the MSM distance computation can be improved so that it is competitive with state-of-the-art algorithms for DTW [5]. For MSM-MEDIAN, the problem of computing a time series that minimizes the sum of the MSM-distances to a given set  $X$  of time series, the running times are even better than for DTW-MEAN<sup>1</sup>: There is a DP algorithm that solves MSM-MEDIAN in  $\mathcal{O}(n^{k+3}2^k k^3)$  time [6].<sup>2</sup> As in the DTW-MEAN algorithm, this is a polynomial running time for fixed number of input sequences but the running time dependence on  $k$  is better. This running time advantage was also confirmed in an experimental evaluation [6]. Given the high quality of MSM-based classifications, it would be very desirable to push this running time advantage even further by finding better algorithms for MSM-MEDIAN. At this point, it should be noted that apart from the above-mentioned DP, nothing is known about the complexity of MSM-MEDIAN. This work aims at filling this gap. In particular, we study whether there is hope for substantial improvements over the current DP algorithm.

**Our Results.** We present two main results. First, we show that MSM-MEDIAN is NP-hard, W[1]-hard when parameterized by  $k$ , and cannot be solved in  $f(k) \cdot |I|^{\mathcal{O}(k)}$  time for any computable function  $f$ , unless the ETH fails. Here,  $|I|$  denotes the total input size of MSM-MEDIAN. These hardness results hold even if the total number of distinct values in the input time series is three. This implies that the previous DP algorithm for MSM-MEDIAN [6]

<sup>1</sup> It may seem inappropriate to directly compare a mean finding problem with a median finding problem. However, in the DTW-MEAN problem, the DTW distance measure is defined as a square root of a warping past cost [2]. Consequently, the DTW mean minimizes the sum of warping path costs and is thus in fact a median for this distance measure.

<sup>2</sup> Previously, the problem was called MSM-MEAN. Following a reviewer suggestion, we now use the name MSM-MEDIAN since the aim is to minimize the sum of distances instead of the sum of squared distances.

is close to optimal. Second, we show that MSM-MEDIAN can be solved in  $2^{\mathcal{O}(d/c)} \cdot |I|^{\mathcal{O}(1)}$  time where  $c$  is the constant cost for a merge or split operation and  $d$  is a bound on the total distance of the median  $x^*$  to the input time series in  $X$ . Due to space constraints, proofs of statements marked with (\*) are deferred to a full version.

## 2 Preliminaries

A *time series* of length  $n$  is a sequence  $x = (x[1], \dots, x[n])$ , where each *data point*, in short *point*,  $x[i]$  is a real number. For  $i \in [1, n]$  and  $j \in [i, n]$ , we denote by  $x[i, j]$  the contiguous subsequence of  $x$  starting at  $i$  and ending at  $j$ . A contiguous subsequence  $x[i, j]$  is an  $\alpha$ -*run* if  $x[\ell] = \alpha$  for each  $\ell \in [i, j]$ . Similarly, we call  $x[i, j]$  a *binary run* if  $x[\ell] \in \{0, 1\}$  for each  $\ell \in [i, j]$ . For a set of time series  $X = \{x_1, \dots, x_k\}$ , the  $i$ th point of the  $j$ th time series of  $X$  is denoted by  $x_j[i]$ ;  $V(X) = \{x[i] \mid x \in X, i \in [1, |x|]\}$  denotes the set of all values of points in the time series of  $X$ .

**Move-Split-Merge Operations.** We now define the MSM metric, following the notation of Stefan et al. [12], and the MSM-MEDIAN problem. The MSM metric allows three types of operations to transfer one time series into another: *move*, *split*, and *merge*. For time series  $x = (x[1], \dots, x[n])$ , a move transforms a point  $x[i]$  into  $x[i] + w$  for some  $w \in \mathbb{R}$ , that is,  $\text{Move}_{i,w}(x) := (x[1], \dots, x[i-1], x[i] + w, x[i+1], \dots, x[n])$ , with cost  $\text{Cost}(\text{Move}_{i,w}) = |w|$ . Informally, we say that there is a *move at point  $x[i]$  to another point  $x[i] + w$* . The split operation splits the  $i$ th element of  $x$  into two consecutive points. A split at point  $x[i]$  is defined as  $\text{Split}_i(x) := (x[1], \dots, x[i-1], x[i], x[i], x[i+1], \dots, x[n])$ . A merge operation may be applied to two consecutive points of equal value. For  $x[i] = x[i+1]$ , it is given by  $\text{Merge}_i(x) := (x[1], \dots, x[i-1], x[i+1], \dots, x[n])$ . We say that  $x[i]$  and  $x[i+1]$  *merge*. Split and merge operations are inverse operations. Their costs are assumed to be equal and determined by a given nonnegative constant  $c =: \text{Cost}(\text{Split}_i) = \text{Cost}(\text{Merge}_i)$ .

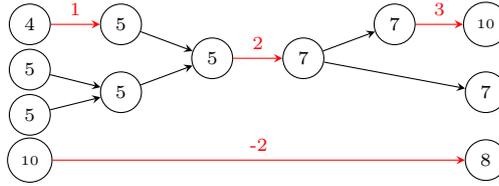
A *transformation sequence*  $\mathbb{S}$  is a tuple  $(S_1, \dots, S_s)$  with  $S_j \in \{\text{Move}_{i_j, w_j}, \text{Split}_{i_j}, \text{Merge}_{i_j}\}$ . A *transformation*  $\mathbf{T}(x, \mathbb{S})$  of a time series  $x$  for a given transformation sequence  $\mathbb{S}$  is defined recursively via  $\mathbf{T}(x, \mathbb{S}) := \mathbf{T}(S_1(x), (S_2, \dots, S_s))$  and  $\mathbf{T}(x, \emptyset) := x$ . We say that  $\mathbb{S}$  transforms  $x$  to  $y$  if  $\mathbf{T}(x, \mathbb{S}) = y$ . The cost of a transformation sequence  $\mathbb{S}$  is the sum of all individual operation costs, that is,  $\text{Cost}(\mathbb{S}) := \sum_{S \in \mathbb{S}} \text{Cost}(S)$ . A transformation is *optimal* if it has minimal cost. The *MSM distance*  $d_{\text{MSM}(c)}(x, y)$  between two time series  $x$  and  $y$  is the cost of an optimal transformation. If  $c$  is clear from context, we may only write  $d_{\text{MSM}}$ . We let  $D_{\text{MSM}(c)}(X, y) := \sum_{x \in X} d_{\text{MSM}(c)}(x, y)$  denote the distance of a sequence  $X$  of time series to a time series  $y$ . A *median*  $x^*$  of a set of time series  $X$  is a time series with minimum distance to  $X$ . The decision problem of computing a median is defined as follows.

MSM-MEDIAN

**Input:** A constant  $c > 0$ , a sequence  $X := (x_1, \dots, x_k)$  of time series, and an integer  $d$ .

**Question:** Is there a time series  $x^*$  such that  $D_{\text{MSM}(c)}(X, x^*) \leq d$ ?

**Transformations Graphs.** We further recall the concept of *transformation graphs* to describe the structure of a transformation between time series  $x$  and  $x^*$  [6, 12]. The *transformation graph* of  $\mathbf{T}(x, \mathbb{S}) = x^*$  is a directed acyclic graph  $G_{\mathbb{S}}(x, x^*)$  with *source nodes*  $N(x) = \{u[1], \dots, u[m]\}$  and *sink nodes*  $N(x^*) = \{u^*[1], \dots, u^*[n]\}$ , where a node  $u[i]$  represents the point  $x[i]$  and a node  $u^*[j]$  represents the point  $x^*[j]$ . The nodes which are neither source nor sink nodes are called *intermediate nodes*. All nodes have in-degree and out-degree at most 2. If there is a directed path from one node  $\alpha$  to another node  $\beta$ , we say that  $\alpha$  is *aligned*



■ **Figure 1** An optimal transformation graph between  $x := (4, 5, 5, 10)$  and  $y := (10, 7, 8)$  with  $c = 0.1$ . Move edges are red. There are two merge operations, one split operation, and move operations of total cost 8. Hence,  $d(x, y) = 8.3$ .

to  $\beta$ . The nodes of  $N(x)$  that align to the same node  $\alpha$  correspond to consecutive points in  $x$ . Each node in the node set  $V$  of  $G_{\mathbb{S}}(x, x^*)$  is associated with a value given by a function  $\text{val} : V \rightarrow \mathbb{R}$ . For source and sink nodes we have  $\text{val}(u[i]) = x[i]$  and  $\text{val}(u^*[j]) = x^*[j]$ . Each intermediate node is also associated with a value. The edges represent the transformation operations of  $\mathbb{S}$ . To create a transformation graph, for each operation in  $\mathbb{S}$ , a respective *move edge* or two *split* edges, or two *merge edges* are added to the graph. If a node  $\alpha$  has outdegree 2 and is connected to a node  $\beta$  by a split edge and  $\beta$  is a child of  $\alpha$ , then there exists a node  $\gamma \neq \beta$  to which  $\alpha$  is connected by a split edge and which is a child of  $\alpha$ . If the nodes  $\alpha$  and  $\beta$  are connected by a merge edge,  $\alpha$  is a parent of  $\beta$ , and  $\beta$  has indegree 2, then there exists a node  $\gamma \neq \alpha$  which is connected to  $\beta$  by a merge edge and is a parent of  $\beta$ . Moreover, for the split and the merge case, it holds that  $\text{val}(\alpha) = \text{val}(\beta) = \text{val}(\gamma)$ . A move edge can be further specified as an *increasing (inc-)* or *decreasing (dec-)* edge if the move operation adds a (not necessarily strictly) positive or negative value to the value of the parent node, respectively.

A *transformation path*, in short *path*, in  $G_{\mathbb{S}}(x, x^*)$  is a directed path from a source node  $u[i] \in N(x)$  to a sink node  $u^*[j] \in N(x^*)$ . A transformation path is *monotonic* if the move edges on this path are only inc- or only dec-edges. A transformation graph is *optimal* if it belongs to an optimal transformation. There exists an optimal transformation graph  $G_{\mathbb{S}}(x, x^*)$  which can be decomposed into a sequence of distinct *trees*  $(T_1, \dots, T_t)$  with the following properties [6]: 1) The sink and source nodes of each tree  $T_i$  form a contiguous subsequence of  $x$  and  $y$ , respectively. 2) For all  $i \in [1, t - 1]$ , the source and sink nodes of  $T_i$  precede the source and sink nodes of  $T_{i+1}$ , respectively. 3) Each path in each  $T_i$  is monotonic. For  $i \in [1, n]$  and  $j \in [i, n]$ , we denote by  $u[i, j]$  a contiguous subsequence of nodes of  $x$ .

The *cost of a tree*  $T$  is the sum of the cost of the tree edges and denoted by  $\text{Cost}_c(T)$ . If the merge/split cost  $c$  is clear from the context, we may omit the subscript. If a tree contains two nodes  $\alpha$  and  $\beta$ , its cost are at least  $|\alpha - \beta|$ . If a tree contains a split or merge edge, its cost are at least  $c$ . We call an optimal transformation graph decomposed into a sequence of trees as an *optimal transformation forest*. Figure 1 shows an example where the transformation graph consists of two trees.

► **Observation 1 (\*)**. *Let  $X$  be a set of time series, let  $x^*$  be a time series, and let  $i \in [1, |x^*| - 1]$ . If for each time series  $x \in X$ , there is some optimal transformation graph containing a tree  $T_x$  such that both  $u^*[i]$  and  $u^*[i + 1]$  are sinks of  $T_x$ , then there is a time series  $y^*$  such that for each time series  $x \in X$ ,  $d_{\text{MSM}(c)}(x, x^*) > d_{\text{MSM}(c)}(x, y^*)$ .*

Let  $x$  and  $y$  be time series of the same length. We define  $d_{\text{Move}}(x, y) := \sum_{i=1}^{|x|} |x[i] - y[i]|$  as the *move distance* between  $x$  and  $y$ . The move distance describes the cost of a transformation forest between  $x$  and  $y$  that only uses move operations. Hence,  $d_{\text{Move}}(x, y) = d_{\text{MSM}(c)}(x, y)$  if and only if some optimal transformation between  $x$  and  $y$  uses only move operations.

**Circular Consensus String.** Our negative results are obtained by a reduction from BINARY CIRCULAR CONSENSUS STRING [4]. Let  $s$  be a string of length  $n$  and let  $\delta \in [1, n]$ . The *circular shift of  $s$  by  $\delta$*  is the string  $s^{\leftarrow\delta}$  of length  $n$  with  $s^{\leftarrow\delta}[i] := s[1 + (i - 1 + \delta) \bmod n]$  for each  $i \in [1, n]$ . We denote the *Hamming distance* of strings  $s_1$  and  $s_2$  by  $d_{\text{Ham}}(s_1, s_2)$ .

BINARY CIRCULAR CONSENSUS STRING

**Input:** A set  $S := \{s_1, \dots, s_k\}$  of binary strings of length  $n$  and an integer  $d$ .

**Question:** Is there a binary string  $s^*$  of length  $n$  and a  $k$ -tuple  $(\delta_1, \dots, \delta_k) \in [0, n-1]^k$  such that  $\sum_{i \in [1, k]} d_{\text{Ham}}(s_i^{\leftarrow\delta_i}, s^*) \leq d$ ?

The Exponential Time Hypothesis (ETH) [7] implies that 3-SAT cannot be solved in  $2^{o(|F|)}$  time where  $F$  is the input formula. Assuming the ETH, BINARY CIRCULAR CONSENSUS STRING cannot be solved in  $f(k) \cdot n^{o(k)}$  time for any computable function  $f$  [4].

### 3 Finding an MSM-Median is Hard

In this section we prove our main hardness results for MSM-MEDIAN.

► **Theorem 2.** *For  $c = 1$ , MSM-MEDIAN is NP-hard, W[1]-hard when parameterized by  $k$ , and cannot be solved in  $f(k) \cdot |I|^{o(k)}$  time for any computable function  $f$ , unless the ETH fails. This holds even if  $|V(X)| = 3$ .*

To show the hardness results, we present a reduction from the BINARY CIRCULAR CONSENSUS STRING-problem which is NP-hard, W[1]-hard when parameterized by  $k$ , and cannot be solved in  $f(k) \cdot n^{o(k)}$  time for any computable function  $f$ , unless the ETH fails [4].

Let  $I := (S, d)$  be an instance of BINARY CIRCULAR CONSENSUS STRING and let  $n$  denote the length of each binary strings of  $S := \{s_1, \dots, s_k\}$  with  $k := |S|$ . If  $d \geq n \cdot k$ , then  $I$  is a trivial yes-instance. Hence, we assume that  $d \leq n \cdot k$ . We can assume that  $k \leq n$  as otherwise  $I$  can be solved in FPT-time for  $k$ . We now describe how to construct in polynomial time an equivalent instance  $I' := (c = 1, X, d')$  of MSM-MEDIAN such that  $|X| = |S|$  and  $|I'| \in n^{O(1)}$ . Each point in each time series of  $X$  has value either 0, 1, or  $A := 2d + 3$ . Let  $g : \{0, 1\}^* \rightarrow \{0, 1, A\}^*$  be the function where  $g(0) := (0, A)$  and  $g(1) := (1, A)$ , and for any binary string  $y$  of length at least 2 we have  $g(y) := g(y[1]) \circ \dots \circ g(y[|y|])$ .

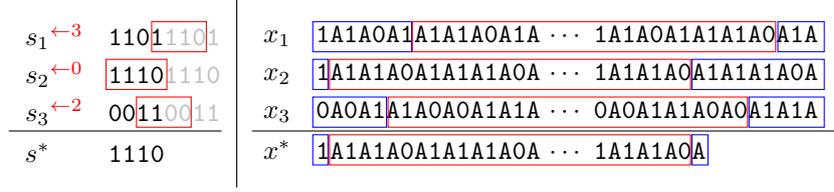
For each  $i \in [1, k]$ , we define a time series  $x'_i := g(s_i)$ . Let  $R := k \cdot (n \cdot (A + 2) + 1)$ . We set  $X := (x_1, \dots, x_k)$ , where for each  $i \in [1, k]$ ,  $x_i$  is the concatenation of  $R$  copies of  $x'_i$ , that is,  $x_i := (x'_i)^R = (g(s_i))^R = g((s_i)^R)$ . Finally, we set  $d' := (R - 1) \cdot d + k \cdot (n \cdot (A + 2) + 1) = (R - 1) \cdot d + k \cdot (n \cdot (2d + 5) + 1)$ . This completes the construction of  $I'$ .

Note that  $|X| = k$  and that  $|I'| \in |I|^{O(1)} \subseteq n^{O(1)}$  since we assumed that  $k \leq n$  and  $d \leq n \cdot k$ . Hence, to show the statement, it remains to show that  $I$  is a yes-instance of BINARY CIRCULAR CONSENSUS STRING if and only if  $I'$  is a yes-instance of MSM-MEDIAN.

**$I'$  is a yes-instance if  $I$  is a yes-instance.** Let  $s^*$  be a binary string of length  $n$  and let  $(\delta_1, \dots, \delta_k) \in [0, n - 1]^k$  be a  $k$ -tuple such that  $\sum_{i \in [1, k]} d_{\text{Ham}}(s_i^{\leftarrow\delta_i}, s^*) \leq d$ . We define a time series  $x'$  as  $x' := g(s^*)$ . Let  $x^*$  be the concatenation of  $R - 1$  copies of  $x'$ , that is,

$$x^* := (x')^{R-1} = (g(s^*))^{R-1} = g((s^*)^{R-1}).$$

We show that  $D_{\text{MSM}}(X, x^*) \leq d'$ . More precisely, we show that  $d_{\text{MSM}}(x_i, x^*) \leq (R - 1) \cdot d_{\text{Ham}}(s_i^{\leftarrow\delta_i}, s^*) + n \cdot (A + 2) + 1$  for each  $i \in [1, k]$ . Since  $\sum_{s_i \in S} d_{\text{Ham}}(s_i^{\leftarrow\delta_i}, s^*) \leq d$  and  $D_{\text{MSM}}(X, x^*) = \sum_{x_i \in X} d_{\text{MSM}}(x_i, x^*)$ , this then implies that  $D_{\text{MSM}}(X, x^*) \leq (R - 1) \cdot d + k \cdot (n \cdot (A + 2) + 1) = d'$ . Informally, we obtain the bound on  $d_{\text{MSM}}(x_i, x^*)$  via the



■ **Figure 2** Left: An instance of CCS for three strings  $s_1, s_2, s_3$  with the consensus string  $s^*$ . Red rectangles show the shift  $\delta_i$ ,  $i \in [1, 3]$ . Right: Corresponding MSM-MEDIAN problem with input time series  $x_1, x_2$ , and  $x_3$  and the median  $x^*$ . Blue parts in the input time series align via merge-and-move edges to  $x^*$ . Red parts align via move edges from the input time series to  $x^*$ .

following transformation: For the middle  $(R-1) \cdot 2 \cdot n$  points of  $x_i$ , only move operations are applied. All other points at the beginning and end of each time series in  $X$  merge to the first or last point in  $x^*$ , respectively. Figure 2 shows an example.

The above-mentioned set of middle points of  $x_i$  is defined as follows. The string  $(s_i)^R$  contains the substring  $(s_i^{\leftarrow \delta_i})^{R-1}$  starting at index  $\delta_i + 1$ . Hence,  $x_i = g((s_i)^R)$  contains the substring  $\tilde{x}_i := g((s_i^{\leftarrow \delta_i})^{R-1}) = g(s_i^{\leftarrow \delta_i})^{R-1}$  starting at index  $2 \cdot \delta_i + 1$ . The time series  $\tilde{x}_i$  comprises exactly these middle points. We now bound the distance of  $x_i$  to  $\tilde{x}_i$  and the distance of  $\tilde{x}_i$  to  $x^*$ .

First, we show that  $d_{\text{MSM}}(x_i, \tilde{x}_i) \leq n \cdot (A+2) + 1$ . To this end, we describe a transformation graph  $G_i$  between  $x_i$  and  $\tilde{x}_i$  that consists of  $(R-1) \cdot 2 \cdot n$  trees. The first tree  $T_1$  contains the first  $2 \cdot \delta_i + 1$  points of  $x_i$  as source nodes and the first point of  $\tilde{x}_i$  as the unique sink node. Similarly, the last tree  $T_{(R-1) \cdot 2 \cdot n}$  contains the last  $2 \cdot (n - \delta_i) + 1$  points of  $x_i$  as source nodes and the last point of  $\tilde{x}_i$  as the unique sink node. For each  $\ell \in [2, (R-1) \cdot 2 \cdot n - 1]$ , the tree  $T_\ell$  consists of a single edge from the source  $u_i[2 \cdot \delta_i + \ell]$  to the sink  $\tilde{u}_i[\ell]$ . Since  $x_i$  contains the substring  $\tilde{x}_i$  starting at index  $2 \cdot \delta_i + 1$ , for each  $\ell \in [2, (R-1) \cdot 2 \cdot n - 1]$ , we have  $x_i[2 \cdot \delta_i + \ell] = \tilde{x}_i[\ell]$  which implies  $\text{Cost}(T_\ell) = 0$ . Hence, it remains to show that  $\text{Cost}(T_1) + \text{Cost}(T_{(R-1) \cdot 2 \cdot n}) \leq n \cdot (A+2) + 1$ . The following lemma upper-bounds the costs of these trees independent of the concrete binary values of their respective sources and sinks. For each  $\alpha \in \{0, 1, A\}$ , we use  $\alpha$  as shortcut for the length-one time series  $(\alpha)$ .

► **Lemma 3 (\*)**. *Let  $y$  be a binary string. It holds that*

- $d_{\text{MSM}}(A \circ g(y), A) \leq |y| \cdot (A+2)$  and
- for each  $\alpha_1 \in \{0, 1\}$  and each  $\alpha_2 \in \{0, 1\}$ ,  $d_{\text{MSM}}(g(y) \circ \alpha_1, \alpha_2) \leq |y| \cdot (A+2) + 1$ .

Recall that sink nodes of  $T_1$  are nodes in  $x^*$ . Since the unique sink node of  $T_1$  has value either 0 or 1, Lemma 3 implies  $\text{Cost}(T_1) \leq (\delta_i) \cdot (A+2) + 1$ . Moreover, since the unique sink node of  $T_{(R-1) \cdot 2 \cdot n}$  has value  $A$ , Lemma 3 implies  $\text{Cost}(T_{(R-1) \cdot 2 \cdot n}) \leq (n - \delta_i) \cdot (A+2)$ . Hence,  $d_{\text{MSM}}(x_i, \tilde{x}_i) \leq \text{Cost}(T_1) + \text{Cost}(T_{(R-1) \cdot 2 \cdot n}) \leq n \cdot (A+2) + 1$ .

Second, we show that  $d_{\text{MSM}}(\tilde{x}_i, x^*) \leq (R-1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*)$ . Recall that  $\tilde{x}_i$  and  $x^*$  have the same length  $((R-1) \cdot 2 \cdot n)$ . Hence, it is sufficient to show that  $d_{\text{Move}}(\tilde{x}_i, x^*) \leq (R-1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*)$ . Since  $\tilde{x}_i[\ell] = x_i^*[\ell] = A$  for each even  $\ell \in [1, |x^*|]$ , we conclude

$$\begin{aligned}
 d_{\text{Move}}(\tilde{x}_i, x^*) &= \sum_{\text{odd } \ell \in [1, |x^*|]} |\tilde{x}_i[\ell] - x^*[\ell]| = \sum_{\ell \in [1, (R-1) \cdot n]} |\tilde{x}_i[2\ell - 1] - x^*[2\ell - 1]| \\
 &= (R-1) \cdot \sum_{\ell \in [1, n]} |\tilde{x}_i[2\ell - 1] - x^*[2\ell - 1]| \\
 &= (R-1) \cdot \sum_{\ell \in [1, n]} d_{\text{Ham}}(\tilde{x}_i[2\ell - 1], x^*[2\ell - 1])
 \end{aligned}$$

$$\begin{aligned}
 &= (R - 1) \cdot \sum_{\ell \in [1, n]} d_{\text{Ham}}(s_i^{\leftarrow \delta_i}[\ell], s^*[\ell]) \\
 &= (R - 1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*).
 \end{aligned}$$

Recall that  $\tilde{x}_i = (g(s_i^{\leftarrow \delta_i}))^{R-1}$  and  $x^* = (g(s^*))^{R-1}$ . The second to last equation holds since by definition of  $g$ , for each  $j \in [1, n]$ ,  $g(s_i^{\leftarrow \delta_i})[2j - 1] = s_i^{\leftarrow \delta_i}[j]$  and  $g(s^*)[2j - 1] = s^*[j]$ . Hence,  $d_{\text{MSM}}(\tilde{x}_i, x^*) \leq d_{\text{Move}}(\tilde{x}_i, x^*) = (R - 1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*)$ .

Since  $d_{\text{MSM}}$  is a metric, we obtain

$$d_{\text{MSM}}(x_i, x^*) \leq d_{\text{MSM}}(x_i, \tilde{x}_i) + d_{\text{MSM}}(\tilde{x}_i, x^*) \leq (R - 1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*) + n \cdot (A + 2) + 1.$$

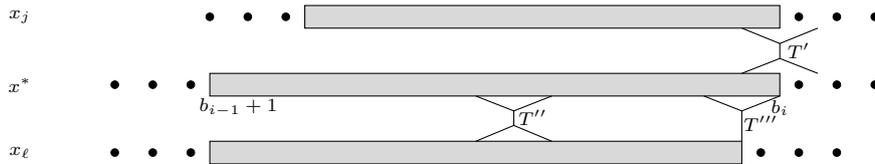
Hence,  $d_{\text{MSM}}(x_i, x^*) \leq (R - 1) \cdot d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*) + n \cdot (A + 2) + 1$  for each time series  $x_i \in X$  and thus  $D_{\text{MSM}}(X, x^*) \leq d'$ . Consequently,  $I'$  is a yes-instance of MSM-MEDIAN.

**$I'$  is a no-instance if  $I$  is a no-instance.** If  $I$  is a no-instance, then for each binary string  $s^*$  of length  $n$  and each  $k$ -tuple  $(\delta_1, \dots, \delta_k) \in [0, n - 1]^k$ ,  $\sum_{i \in [1, k]} d_{\text{Ham}}(s_i^{\leftarrow \delta_i}, s^*) \geq d + 1$ . Let  $x^*$  be a time series that minimizes  $D_{\text{MSM}}(X, x^*)$ . We show that  $D_{\text{MSM}}(X, x^*) \geq R \cdot (d + 1) = d' + d > d$ . We can assume that  $x^*$  uses only values of  $V(X) = \{0, 1, A\}$  [6]. For each  $i \in [1, k]$ , let  $G_{S_i}(x_i, x^*)$  be an optimal transformation graph between  $x_i$  and  $x^*$ . Moreover, let  $\mathcal{T}_i$  be the collection of all trees of  $G_{S_i}(x_i, x^*)$ . We can assume that each value in each such tree is from  $V(X) = \{0, 1, A\}$  [6]. For a collection of trees  $\mathfrak{T}$ , we denote  $\text{Cost}(\mathfrak{T}) := \sum_{T \in \mathfrak{T}} \text{Cost}(T)$ . To show that  $D_{\text{MSM}}(X, x^*) > d'$ , we first introduce some notation.

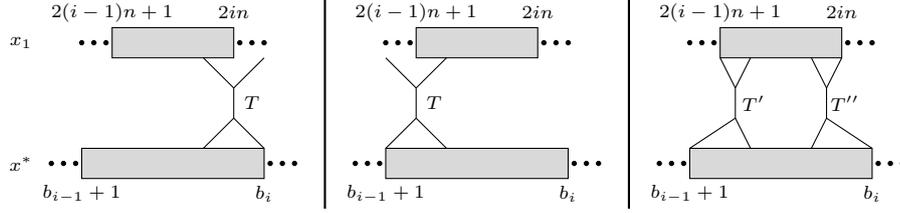
We say that a move edge  $(u_1, u_2)$  of any tree is *heavy* if  $|\text{val}(u_1) - \text{val}(u_2)| > 1$ . Analogously, we call path  $P$  in any tree *heavy* if at least one move edge of  $P$  is heavy. Since each node of each tree between  $X$  and  $x^*$  has a value from  $\{0, 1, A\}$ , a tree  $T$  contains a heavy path if and only if  $T$  contains at least one node with value  $A$  and at least one node with value either 0 or 1. In other words, if a tree  $T$  contains no heavy path, then a) the value of each node in  $T$  is  $A$  or b) the value of each node in  $T$  is from  $\{0, 1\}$ . Since  $A = 2d + 3$ , the cost of a tree with a heavy path is at least  $A - 1 = 2(d + 1)$ .

In the following, we take the time series  $x_1 \in X$  as a pivot and regard the partial alignment of a prefix of  $x_1$  to a prefix of  $x^*$ . Then, we analyze the cost of all other time series  $x \in X \setminus \{x_1\}$  aligning to this prefix of  $x^*$ .

For each  $i \in [0, R]$ , let  $b_i$  be the largest number of  $[0, |x^*|]$  such that  $u^*[b_i]$  is the sink of a tree of  $\mathcal{T}_1$  containing no source nodes of  $u_1[2ni + 1, 2nR]$ . Note that  $b_0 = 0$  and that if the tree of  $\mathcal{T}_1$  that has  $u_1[1]$  as a source node also has a source node  $u_1[2ni + 1]$  for some  $i \in [1, R - 1]$ , then  $b_j = 0$  for each  $j \in [0, i]$ . For each  $i \in [0, R]$  let  $\mathcal{B}_i$  be set of trees between any time series  $x$  of  $X$  and  $x^*$  containing only sinks of  $u^*[1, b_i]$ . Figure 3 depicts examples of trees belonging to  $\mathcal{B}_i$  and a tree not belonging to  $\mathcal{B}_i$ . For  $i \in [1, R]$ , the  $i$ th block is defined as  $\mathcal{Q}_i = \mathcal{B}_i \setminus \mathcal{B}_{i-1}$ . That is, each tree  $T$  in a block  $\mathcal{Q}_i$  contains only sinks of  $u^*[1, b_i]$  and at least one sink of  $u^*[b_{i-1} + 1, b_i]$ . The  $i$ th block  $\mathcal{Q}_i$  is a *cut* if  $\mathcal{Q}_i \cap \mathcal{T}_1$  has exactly the set of source nodes  $u_1[2n(i - 1) + 1, 2ni]$ . Figure 4 depicts two examples



■ **Figure 3** Two time series  $x_j$  and  $x_l$  with a median  $x^*$  of  $X$ ;  $T'$  is not in  $\mathcal{B}_i$ ,  $T''$  and  $T'''$  are in  $\mathcal{B}_i$ .



■ **Figure 4** The upper time series shows  $x_1$ , the lower time series shows the median  $x^*$ . The first example is not a cut since the tree  $T$  has source nodes  $u_1[j]$  with  $j > 2in$ . The second example is not a cut since the tree  $T$  has source nodes  $u_1[j]$  with  $j \leq 2(i-1)n$ . The third example is a cut.

of blocks not being cuts and one example of a cut. The idea behind the definitions of cuts is as follows: If a block  $\mathcal{Q}_i$  is not a cut, then some tree  $T \in \mathcal{T}_1$  with at least two sources contains  $u_1[2n(i-1)]$  or  $u_1[2ni+1]$  as a source. We say that  $\mathcal{Q}_i$  is a *light cut* if for each tree  $T \in \mathcal{Q}_i \cap \mathcal{T}_1$ ,  $T$  contains no heavy path. Hence, if  $\mathcal{Q}_i$  is a light cut, then for each source node  $\hat{u}$  in  $u_1[2n(i-1)+1, 2ni]$ , the tree of  $\mathcal{T}_1$  containing  $\hat{u}$  contains no heavy path. Note that a light cut  $\mathcal{Q}_i$  may still contain trees with heavy paths but these trees are not contained in  $\mathcal{T}_1$ .

We further describe the structure of a light cut. By construction, every copy of  $x_1$  starts with a binary value followed by an  $A$ , followed by a binary value and so on. All paths in trees of a light cut  $\mathcal{Q}_i$  are light. That is, an  $A$  in  $x_1$  aligns to one or multiple  $A$  in  $x^*$  and a binary number in  $x_1$  aligns to one or multiple binary numbers in  $x^*$ . That is, we have  $n$  non-empty binary runs  $(r_{\text{bin}}^1, \dots, r_{\text{bin}}^n)$  and  $n$  non-empty  $A$ -runs  $(r_A^1, \dots, r_A^n)$  such that

$$x^*[b_{i-1}+1, b_i] = (r_{\text{bin}}^1 \circ r_A^1 \circ r_{\text{bin}}^2 \circ r_A^2 \circ \dots \circ r_{\text{bin}}^n \circ r_A^n).$$

We now show that each block  $\mathcal{Q}_i$  has amortized cost at least  $d+1$ . This implies that the total cost of the transformation forest exceeds  $d' < R \cdot (d+1)$  and thus  $I'$  is a no-instance of MSM-MEDIAN. Let  $i \in [1, R]$ . We say that a set  $J \subseteq [i, R]$  with  $i \in J$  is *right-dominated* by a set  $J' \subseteq [i, R]$  if  $J \subseteq J'$  and  $[\max(J), \max(J')] \subseteq J'$ .

► **Lemma 4 (\*)**. *Let  $i \in [1, R]$  such that the block  $\mathcal{Q}_i$  is not a light cut. Moreover, let  $J \subseteq [i, R]$  such that  $i \in J$  and for each  $j \in J$ , the block  $\mathcal{Q}_j$  is not a light cut. Then, there is some  $J' \subseteq [i, R]$  such that  $J$  is right-dominated by  $J'$  and  $\text{Cost}(\cup_{j \in J'} \mathcal{Q}_j \cap \mathcal{T}_1) \geq |J'| \cdot (d+1)$ .*

► **Lemma 5**. *Let  $I$  be a no-instance of BINARY CIRCULAR CONSENSUS STRING and let  $i \in [0, R-1]$ . If  $\text{Cost}(\mathcal{B}_i) \geq i \cdot (d+1)$ , then there is some  $j > i$  such that  $\text{Cost}(\mathcal{B}_j) \geq j \cdot (d+1)$ .*

**Proof.** Considering the next block  $\mathcal{Q}_{i+1}$ , we distinguish whether  $\mathcal{Q}_{i+1}$  is a light cut.

**Case 1: Block  $\mathcal{Q}_{i+1}$  is not a light cut.** Let  $J = \{i+1\}$ . By Lemma 4, there is some  $J' \subseteq [i+1, R]$  with  $J \subseteq J'$  and  $[\max(J), \max(J')] = [i+1, \max(J')] \subseteq J'$  such that  $\text{Cost}(\cup_{j \in J'} \mathcal{Q}_j \cap \mathcal{T}_1) \geq |J'| \cdot (d+1)$ . For  $j = \max(J')$  we have  $\text{Cost}(\mathcal{B}_j \setminus \mathcal{B}_i) \geq \text{Cost}(\cup_{j \in J'} \mathcal{Q}_j \cap \mathcal{T}_1) \geq |J'| \cdot (d+1) = (j-i) \cdot (d+1)$ . We get

$$\begin{aligned} \text{Cost}(\mathcal{B}_j) &= \text{Cost}((\mathcal{B}_j \setminus \mathcal{B}_i) \cup \mathcal{B}_i) = \text{Cost}(\mathcal{B}_j \setminus \mathcal{B}_i) + \text{Cost}(\mathcal{B}_i) \\ &\geq (j-i) \cdot (d+1) + i \cdot (d+1) = j \cdot (d+1). \end{aligned}$$

Hence, the statement holds for  $j$ .

**Case 2: Block  $\mathcal{Q}_{i+1}$  is a light cut.** Recall that since  $\mathcal{Q}_{i+1}$  is a light cut, there are  $n$  non-empty binary runs  $(r_{\text{bin}}^1, \dots, r_{\text{bin}}^n)$  and  $n$  non-empty  $A$ -runs  $(r_A^1, \dots, r_A^n)$  such that

$$x^*[b_i+1, b_{i+1}] = (r_{\text{bin}}^1 \circ r_A^1 \circ r_{\text{bin}}^2 \circ r_A^2 \circ \dots \circ r_{\text{bin}}^n \circ r_A^n).$$

First, we show two properties of transformation graphs from each time series of  $X$  to  $x^*$  which directly imply  $\text{Cost}(\mathcal{B}_j) \geq j \cdot (d + 1)$  for some  $j \geq i + 1$ . Afterward, we show that transformation graphs without these properties then resembles circular shifts of the binary input strings of  $I$ . Because  $I$  is a no-instance of BINARY CIRCULAR CONSENSUS STRING, we then get  $\text{Cost}(\mathcal{Q}_{i+1}) \geq d + 1$  which implies  $\text{Cost}(\mathcal{B}_{i+1}) \geq (i + 1) \cdot (d + 1)$ .

▷ **Claim 6 (\*)**. Let  $i \in [0, R - 1]$  such that  $\mathcal{Q}_{i+1}$  is a light cut. If there is some tree  $T$  in  $\mathcal{B}_R \setminus \mathcal{T}_1$  that contains at least one sink node of  $u^*[b_i + 1, b_{i+1}]$  and where any of the following holds:

1.  $T$  is contained in  $\mathcal{Q}_{i+1}$  and contains a heavy path or
  2. the values of the sinks of  $T$  contain at least one  $A$  and at least one binary value,
- then there is some  $j > i$  such that  $\text{Cost}(\mathcal{B}_j \setminus \mathcal{B}_i) \geq (j - i) \cdot (d + 1)$ .

Hence, if Condition 1 or Condition 2 holds, then there is some  $j \geq i + 1$  such that  $\text{Cost}(\mathcal{B}_j \setminus \mathcal{B}_i) \geq (j - i) \cdot (d + 1)$ . Consequently,  $\text{Cost}(\mathcal{B}_j) = \text{Cost}(\mathcal{B}_j \setminus \mathcal{B}_i) + \text{Cost}(\mathcal{B}_i) \geq j \cdot (d + 1)$  which implies that the statement holds for  $j$ . In the following, we thus assume that neither Condition 1 or Condition 2 holds. This implies that

- each tree of  $\mathcal{Q}_{i+1}$  has only one source node (otherwise, Condition 1 holds) and
- each tree with at least one sink node of  $x^*[b_i + 1, b_{i+1}]$  has either a) only sinks with binary values or b) only sinks with value  $A$  (otherwise, Condition 2 holds).

Note that the latter implies that each tree with at least one sink node of  $x^*[b_i + 1, b_{i+1}]$  has either a) only sinks with binary values or b) only sinks with value  $A$ . Since Condition 1 does not hold, this further implies that each tree  $T$  of  $\mathcal{Q}_{i+1}$

- has only sinks with binary values, if the value of the unique source of  $T$  is binary and
- has only sinks with value  $A$ , if the unique source of  $T$  has value  $A$ .

Next, we show that the statement holds for  $j = i + 1$ . To this end, we show that  $\text{Cost}(\mathcal{Q}_{i+1}) \geq d + 1$ . For each  $q \in [1, n]$ , let  $\text{last}^q$  denote the index of the last binary value of  $r_{\text{bin}}^q$  in the median. Let  $s^*$  be the length- $n$  string such that  $s^*[q] = x^*[\text{last}^q]$  for each  $q \in [1, n]$ .

▷ **Claim 7**. For each  $p \in [1, k]$ , there is some even index  $\delta'_p$  such that for each  $q \in [1, n]$  there is a tree  $T_p^q$  satisfying the following properties:

- (i)  $T_p^q$  is contained in  $\mathcal{Q}_{i+1} \cap \mathcal{T}_p$ ,
- (ii)  $u_p[\delta'_p + 2 \cdot q - 1]$  is the unique source node of  $T_p^q$ , and
- (iii)  $u^*[\text{last}^q]$  is a sink node of  $T_p^q$ .

*Proof.* Let  $p \in [1, k]$ . We prove this statement in an inductive way. First, we show that there is an even index  $\delta'_p$  such that there is a tree  $T_p^n$  satisfying Properties (i)–(iii). Afterward, we show that if for some  $\ell \in [2, n]$ , there is a tree  $T_p^\ell$  satisfying Properties (i)–(iii), then the tree  $T_p^{\ell-1}$  satisfies Properties (i)–(iii). This then implies the statement.

Let  $T_p^n$  be the tree of  $\mathcal{T}_p$  having  $u^*[\text{last}^n]$  as a sink node and let  $T'$  be the tree of  $\mathcal{T}_p$  having  $u^*[\text{last}^n + 1]$  as a sink node. Note that  $T_p^n$  satisfies Property (iii). Since  $\text{last}^q$  is the index of the last value of the  $n$ th binary run of  $\mathcal{Q}_{i+1}$ , we have  $x^*[\text{last}^n + 1] = A$ . Moreover, since each tree containing at least one sink node in  $x^*[b_i + 1, b_{i+1}]$  has either a) only sinks with binary values or b) only sinks with value  $A$ ,  $T_p^n$  and  $T'$  are distinct trees. Hence, since  $\text{last}^n + 1 \leq b_{i+1}$ ,  $T_p^n$  is contained in  $\mathcal{Q}_{i+1}$ . This implies that  $T_p^n$  satisfies Property (i). Moreover, since no tree in  $\mathcal{Q}_{i+1}$  contains a heavy path, the values of the source nodes and the values of the sink nodes of  $T_p^n$  are all binary values. Hence, since each tree of  $\mathcal{Q}_{i+1}$  has only one source node,  $T_p^n$  has a unique source and this unique source has a binary value. Since  $x_p$  contains binary values only on odd positions, there is some even  $\delta'_p$  such that  $u_p[\delta'_p + 2 \cdot n - 1]$  is the unique source of  $T_p^n$ . Hence,  $T_p^n$  satisfies Properties (i)–(iii) for  $\delta'_p$ .

Now assume by induction that the statement holds for  $T_p^\ell$ . We show that the tree  $T_p^{\ell-1}$  satisfies Properties (i)–(iii) as well. Since  $r_A^{\ell-1}$  is a non-empty  $A$ -run and each tree of  $\mathcal{Q}_{i+1}$  has either a) only sinks with binary values or b) only sinks with value  $A$ ,  $T_p^\ell$  is not the first tree of  $\mathcal{T}_p$ . Hence, since  $\delta'_p$  is even,  $\delta'_p + 2 \cdot \ell - 1 \geq 3$ . Let  $T'$  be the tree of  $\mathcal{T}_p$  having  $u_p[\delta'_p + 2 \cdot \ell - 2]$  as a source node, and let  $T_p^{\ell-1}$  be the tree of  $\mathcal{T}_p$  having  $u_p[\delta'_p + 2 \cdot \ell - 3] = u_p[\delta'_p + 2 \cdot (\ell - 1) - 1]$  as a source node. Since no tree of  $\mathcal{Q}_{i+1}$  contains a heavy path, the values of all sink nodes of  $T_p^\ell$  and  $T_p^{\ell-1}$  are binary and the values of all sink nodes of  $T'$  are  $A$ . Hence,  $T_p^\ell$  contains exactly the  $\ell$ th binary run as sink nodes, as otherwise,  $T_p^\ell$  contains a sink with value  $A$  or  $T'$  contains a sink with a binary value. Hence,  $T'$  contains the node of the last  $A$  of the  $(\ell - 1)$ th  $A$ -run as a sink node. Since  $T_p^\ell$  is contained in  $\mathcal{Q}_{i+1}$ , this further implies that  $T'$  is also contained in  $\mathcal{Q}_{i+1}$ . Thus, similarly to the above,  $T'$  contains exactly the  $(\ell - 1)$ th  $A$ -run as sink nodes, as otherwise,  $T'$  contains a sink with binary value or  $T_p^{\ell-1}$  contains a sink with value  $A$ . Hence,  $T_p^{\ell-1}$  contains the node  $u^*[\text{last}^{\ell-1}]$  as a sink node and thus fulfills Property (iii). Since  $T_p^\ell$  is contained in  $\mathcal{Q}_{i+1}$ , this further implies that  $T_p^{\ell-1}$  is also contained in  $\mathcal{Q}_{i+1}$ . Hence,  $T_p^{\ell-1}$  fulfills Property (i). By the fact that each tree in  $\mathcal{Q}_{i+1}$  contains only one source node, this then implies that  $u_p[\delta'_p + 2 \cdot (\ell - 1) - 1]$  is the unique source node of  $T_p^{\ell-1}$ . Hence,  $T_p^{\ell-1}$  satisfies Properties (i)–(iii).

Moreover, the above proof also shows that  $\delta'_p + 2 \cdot \ell - 1 \geq 3$  for each  $\ell \in [2, n]$ , which implies that  $\delta'_p \geq 0$ . Additionally, the proof also shows that for each  $\ell \in [2, n]$ ,

- $T_p^\ell$  of  $\mathcal{T}_p$  contains exactly the  $\ell$ th binary run as sink nodes and
- the tree of  $\mathcal{T}_p$  containing  $u_p[\delta'_p + 2 \cdot \ell - 2]$  as unique source node, contains exactly the  $(\ell - 1)$ th  $A$ -run as sink nodes.<sup>3</sup>  $\triangleleft$

For each  $p \in [1, k]$ , let  $\delta'_p$  be the index fulfilling the properties of Claim 7. Moreover, for each  $p \in [1, k]$  and for each  $q \in [1, n]$ , let  $T_p^q$  be the tree fulfilling the properties of Claim 7 with respect to  $\delta'_p$ . Finally, let  $\mathfrak{T} := \{T_p^q \mid p \in [1, k], q \in [1, n]\}$  denote the set of these trees. We show that  $\text{Cost}(\mathfrak{T}) \geq d + 1$ . Due to Property (i) of Claim 7,  $\mathfrak{T} \subseteq \mathcal{Q}_{i+1}$ . Hence,

$$\begin{aligned} \text{Cost}(\mathcal{B}_{i+1}) &= \text{Cost}((\mathcal{B}_{i+1} \setminus \mathcal{B}_i) \cup \mathcal{B}_i) = \text{Cost}(\mathcal{Q}_{i+1}) + \text{Cost}(\mathcal{B}_i) \\ &\geq (d + 1) + i \cdot (d + 1) \geq (i + 1) \cdot (d + 1). \end{aligned}$$

To show that  $\text{Cost}(\mathfrak{T}) \geq d + 1$ , we use the fact that for each binary string  $\hat{s}$  of length  $n$  and each  $k$ -tuple  $(\delta_1, \dots, \delta_k)$ ,  $\sum_{p \in [1, k]} d_{\text{Ham}}(s_p^{\leftarrow \delta_p}, \hat{s}) \geq d + 1$ . In particular, this holds for  $s^*$ , the string of length  $n$  where for each index  $q \in [1, n]$ ,  $s^*[q] = x^*[\text{last}^q]$ . For each  $p \in [1, k]$ , we set  $\delta_p := \frac{\delta'_p \bmod (2n)}{2} = \frac{\delta'_p}{2} \bmod n$ . Next, we show that for each  $p \in [1, k]$ ,  $\text{Cost}(\mathfrak{T}_p) \geq d_{\text{Ham}}(s_p^{\leftarrow \delta_p}, s^*)$ , where  $\mathfrak{T}_p := \mathfrak{T} \cap \mathcal{T}_p = \{T_p^q \mid q \in [1, n]\}$ .

Let  $p \in [1, k]$ . Recall that  $x_p = (g(s_p))^R = g((s_p)^R)$ . Hence, by definition of  $g$  and  $s_p^{\leftarrow \delta_p}$ , for each  $q \in [1, n]$ ,

$$\begin{aligned} s_p^{\leftarrow \delta_p}[q] &= s_p[1 + (\delta_p + q - 1) \bmod n] = (s_p)^R[\delta_p + q] \\ &= x_p[2 \cdot (\delta_p + q) - 1] = x_p[2 \cdot \delta_p + 2 \cdot q - 1] = x_p[\delta'_p + 2 \cdot q - 1]. \end{aligned}$$

Since for each  $q \in [1, n]$ ,  $T_p^q$  contains the source  $u_p[\delta'_p + 2 \cdot q - 1]$  of value  $x_p[\delta'_p + 2 \cdot q - 1] = s_p^{\leftarrow \delta_p}[q]$  and the sink  $u^*[\text{last}^q]$  of value  $s^*[q]$ , we conclude  $\text{Cost}(T_p^q) \geq |s_p^{\leftarrow \delta_p}[q] - s^*[q]| \geq d_{\text{Ham}}(s_p^{\leftarrow \delta_p}[q], s^*[q])$ . Hence,  $\text{Cost}(\mathfrak{T}_p) = \sum_{q=1}^n \text{Cost}(T_p^q) \geq \sum_{q=1}^n d_{\text{Ham}}(s_p^{\leftarrow \delta_p}[q], s^*[q]) = d_{\text{Ham}}(s_p^{\leftarrow \delta_p}, s^*)$ .

Since  $\sum_{p=1}^k d_{\text{Ham}}(s_p^{\leftarrow \delta_p}, s^*) \geq d + 1$ , we conclude  $\text{Cost}(\mathcal{Q}_{i+1}) \geq \text{Cost}(\mathfrak{T}) \geq d + 1$ . This then implies  $\text{Cost}(\mathcal{B}_{i+1}) \geq (i + 1) \cdot (d + 1)$ . Hence, the statement holds for  $j = i + 1$ .  $\blacktriangleleft$

<sup>3</sup> Recall that  $x^*$  minimizes  $D_{\text{MSM}}(X, x^*)$  and that for each  $i \in [1, k]$ ,  $G_i$  is a transformation graph between  $x_i$  and  $x^*$ . By Observation 1, this implies that for each  $\ell \in [2, n]$ , the  $\ell$ th binary run and the  $(\ell - 1)$ th  $A$ -run each have length 1. This then implies  $(x^*[\text{last}^1], \dots, x^*[\text{last}^n + 1]) = g(s^*)$ .

Note that  $\text{Cost}(\mathcal{B}_0) = \text{Cost}(\emptyset) = 0 \geq 0 \cdot (d+1)$ . Hence, due to Lemma 5, one can show via induction that  $\text{Cost}(\mathcal{B}_R) \geq R \cdot (d+1)$ . Since  $R = k \cdot (n \cdot (A+2) + 1)$ ,

$$R \cdot (d+1) = (R-1) \cdot d + R + d = (R-1) \cdot d + k \cdot (n \cdot (A+2) + 1) + d = d' + d.$$

Hence  $D_{\text{MSM}}(X, x^*) = \text{Cost}(\mathcal{B}_R) \geq R \cdot (d+1) > d'$  and  $I'$  is a no-instance of MSM-MEDIAN. This completes the proof of the equivalence of  $I$  and  $I'$  and thus the proof of Theorem 2. With the hardness for  $c = 1$  at hand, we may also show hardness for arbitrary values of  $c$ .

► **Theorem 8 (\*)**. *For every constant  $c > 0$ , MSM-MEDIAN is NP-hard, W[1]-hard when parameterized by  $k$ , and cannot be solved in  $f(k) \cdot |I|^{o(k)}$  time for any computable function  $f$ , unless the ETH fails. This holds even if  $|V(X)| = 3$ .*

## 4 Parameterized Algorithms for MSM-Median

The algorithms presented in the following extend the DP algorithm of Holznigenkemper et al. [6]. Given a sequence  $X$  of time series of length at most  $n$  each and some  $m \in \mathbb{N}$ , this DP computes in time  $\mathcal{O}(n^{|X|+2} \cdot 2^{|X|} \cdot |X|^2 \cdot m)$  a time series  $x^*$  of length at most  $m$  that contains only points of  $V(X)$  and has minimum distance to  $X$  among all such time series.

**Allowing Weights.** Let  $X := (x_1, \dots, x_k)$  be a sequence of time series. Moreover, let  $X' := \{x_i \mid 1 \leq i \leq k\}$  be the set of time series of  $X$  and let  $\omega : X' \rightarrow \mathbb{N}^+$  be the function where for each  $x \in X'$ ,  $\omega(x)$  is the number of occurrences of  $x$  in  $X$ . We call  $(X', \omega)$  the *weighted equivalent* of  $X$ . We denote by  $D_{\text{MSM}(c)}^\omega(X', y) := \sum_{x \in X'} (\omega(x) \cdot d_{\text{MSM}(c)}(x, y))$  the *weighted MSM-distance* between  $X'$  and  $y$ .

► **Observation 9.** *Let  $X := (x_1, \dots, x_k)$  be a sequence of time series and let  $(X', \omega)$  be the weighted equivalent of  $X$ . Then, for each time series  $x^*$ ,  $D_{\text{MSM}(c)}(X, x^*) = D_{\text{MSM}(c)}^\omega(X', x^*)$ .*

► **Lemma 10.** *Let  $X$  be a set of time series of length at most  $n$  each, let  $\omega : X \rightarrow \mathbb{N}^+$  be a weight function, and let  $m \in \mathbb{N}$ . In time  $\mathcal{O}(n^{|X|+2} \cdot 2^{|X|} \cdot |X|^2 \cdot m)$  one can find a time series  $x^*$  that contains only points of  $V(X)$ , has length at most  $m$ , and minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*)$ .*

**Proof.** We adapt the algorithm by Holznigenkemper et al. [6] by inserting the weights of the time series as follows: We fill a  $(k+2)$ -dimensional table  $D$  with entries  $D[\mathbf{p}, \ell, s]$ , where  $\mathbf{p} = (p_1, \dots, p_k)$  indicates the *current positions* of  $X$ , the index  $\ell \in [1, m]$  indicates the *current position* of  $x^*$ , and  $s$  is a point in  $V(X)$ . The entry  $D[\mathbf{p}, \ell, s]$  stores the minimal cost needed to transform the partial time series  $(x_1[1, p_1], \dots, x_k[1, p_k])$  to any time series  $x^*$  of length exactly  $\ell$  where  $x^*[\ell] = s$  and  $x^*$  uses only values from  $V(X)$ . Since all time series are weighted, all partial time series are also weighted. Hence, the cost of all transformation operations regarding the weighted partial time series also have to be weighted.

In the DP recurrence, we distinguish two cases: Merges are applied (to the last positions of a subset of  $X$ ) or splits or moves are applied (to the last positions of all time series in  $X$ ). When merges are applied, the position of  $x^*$  does not change in the recurrences, this case is denoted by  $A_{ME}$ . When moves and splits are applied, then the position of  $x^*$  decreases, this case is denoted  $A_{MS}$ . In the recurrence, we consider the best of these two cases:

$$D[\mathbf{p}, \ell, s] = \min\{A_{MS}[\mathbf{p}, \ell, s], A_{ME}[\mathbf{p}, \ell, s]\}.$$

To present the recurrence for the two cases, we use index sets  $I_{MO}$ ,  $I_{SP}$ , and  $I_{ME}$  representing the time series indices for which move, split, and merge operations are applied, respectively.

## 54:12 On the Complexity of Computing Time Series Medians Under the MSM Metric

All index sets are subsets of  $[1, k]$  and  $I_{MO} \cup I_{SP} = [1, k]$ . For an index set  $I$ , let  $\bar{\mathbf{p}}_I = (\bar{p}_1, \dots, \bar{p}_k)$  where  $\bar{p}_i = p_i - 1$  for all  $i \in I$  and  $\bar{p}_i = p_i$  for all  $i \in [1, k] \setminus I$ . For merge operations, the recursive call of the function does not decrease the current position of  $x^*$ :

$$A_{ME}[\mathbf{p}, \ell, s] = \min_{I_{ME}} (D[\bar{\mathbf{p}}_{I_{ME}}, \ell, s] + \sum_{i \in I_{ME}} (\omega(x_i) \cdot C(x_i[p_i], x_i[p_i - 1], s))),$$

$$\text{where } C(u, v, w) = \begin{cases} c & \text{if } v \leq u \leq w \text{ or } v \geq u \geq w \\ c + \min(|u - v|, |u - w|) & \text{otherwise.} \end{cases}$$

For move and split operations, the recursive call of the function decreases the current position of  $x^*$ :

$$A_{MS}[\mathbf{p}, \ell, s] = \min_{s' \in V(X)} \left\{ \min_{I_{MO}, I_{SP}} (D[\bar{\mathbf{p}}_{I_{MO}}, \ell - 1, s']) + \sum_{i \in I_{MO}} (\omega(x_i) \cdot |x_i[p_i] - s|) + \sum_{i \in I_{SP}} (\omega(x_i) \cdot C(s, x_i[p_i], s')) \right\}.$$

Each single entry of  $A_{MS}$  and  $A_{ME}$  can be computed in time  $2^{|X|} \cdot (|X| + n + d/c + m)^{\mathcal{O}(1)}$ . For the last recursion step, the entries  $D[(1, \dots, 1), 1, s]$  are computed by  $D[(1, \dots, 1), 1, s] = \sum_{i=1}^k (\omega(x_i) \cdot |x_i[1] - s|)$ . All entries  $D[\mathbf{p}, \ell, s]$  for which  $p_i < 1$  for some  $i \in [1, k]$  are set to  $+\infty$ . If  $\ell = 1$  and  $p_i > 1$  for all  $i \in [1, k]$ , then only merge operations may be applied since the position of  $x^*$  can not be decreased anymore:  $D[\mathbf{p}, 1, s] = A_{ME}[\mathbf{p}, \ell, s]$ .

The minimum distance  $D_{\text{MSM}(c)}^\omega(X', x^*)$  to any time series  $x^*$  of length at most  $m$  that uses only points of  $V(X)$  can be computed by  $\min_{\ell \in [m], s \in V(X)} (D[\mathbf{p}, \ell, s])$ , where  $\mathbf{p} := (|x_1|, \dots, |x_k|)$ . Since the previous DP [6] runs in the desired running time and we only added weights to the DP but did not change the table structure, the running time stays the same. The corresponding time series can be found via traceback.  $\blacktriangleleft$

► **Lemma 11 (\*)**. *Let  $X$  be a set of time series of length at most  $n$  each and let  $\omega : X \rightarrow \mathbb{N}^+$ . Then, each time series  $x^*$  that minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*)$  has length at most  $n \cdot |X|$ .*

**Improving the Dynamic Program with a Cost-Bound.** Next, we establish an intermediate FPT algorithm with the parameters  $|X|$  and  $d/c$ .

► **Theorem 12.** *Let  $X$  be a set of time series each of length at most  $n$ , let  $\omega : X \rightarrow \mathbb{N}^+$  be a weight function, let  $d \in \mathbb{R}$ , and let  $m \in \mathbb{N}$ . In time  $4^{|X|} \cdot 3^{d/c} \cdot (|X| + n + d/c + m)^{\mathcal{O}(1)}$  one can find a time series  $x^*$  that contains only points of  $V(X)$ , has length at most  $m$ , and minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*)$  or correctly output that there is no such time series  $x^*$  with  $D_{\text{MSM}(c)}^\omega(X, x^*) \leq d$ .*

The main idea is that when given a cost budget  $d$ , we may be able to limit the number of entries in the DP-table that do not exceed a cost of  $d$ . That is, we only need to compute the entries in the DP-table that are close to the diagonal.

► **Lemma 13 (\*)**. *Let  $\mathbf{p} = (p_1, \dots, p_k) \in \mathbb{N}^k$ , let  $\ell \in \mathbb{N}$ , and let  $s \in \mathbb{R}$ . If  $\sum_{i=1}^k |p_i - \ell| > d/c$ , then  $D[\mathbf{p}, \ell, s] > d$ .*

We now adapt the DP-table described in the proof of Lemma 10 as follows. The table entries now have a slightly different interpretation: If the minimal cost needed to transform the partial time series  $(x_1[1, p_1], \dots, x_k[1, p_k])$  to any time series  $x^*$  of length  $\ell$  with  $x^*[\ell] = s$  that only uses points from  $V(X)$  is at most  $d$ , then the value of  $D[\mathbf{p}, \ell, s]$  is exactly this number. Otherwise,  $D[\mathbf{p}, \ell, s]$  may hold an arbitrary value larger than  $d$ , for example  $d + 1$ .

For each  $\ell \in [1, m]$  and each  $s \in V(X)$ , we only compute the table entries  $D[(p_1, \dots, p_k), \ell, s]$  with  $\sum_{i=1}^k |p_i - \ell| \leq d/c$ . Moreover, whenever a sum required to compute an entry of  $D[\mathbf{p}', \ell', s']$  depends on the value of an entry  $D[\mathbf{p}, \ell, s]$  with  $\sum_{i=1}^k |p_i - \ell| > d/c$ , then this sum is not computed but replaced by  $d+1$  since the sum is at least  $D[\mathbf{p}, \ell, s] > d$  as well. This is correct since each entry of  $D$  is obtained by minimizing sums of non-negative numbers. Finally, we compute  $d^* := \min_{\ell \in [1, m]} \min_{s \in V(X)} D[(|x_1|, \dots, |x_k|), \ell, s]$ . If  $d^* > d$ , we output that there is no time series  $x^*$  with the desired properties. Otherwise, we find some time series  $x^*$  with  $D_{\text{MSM}(c)}^\omega(X, x^*) = d^*$  via traceback and output this time series.

To show that this modified algorithm has the running time promised in Theorem 12, we bound the number of vectors  $(p_1, \dots, p_k) \in \mathbb{N}^k$  with  $\sum_{i=1}^k |p_i - \ell| \leq d/c$ .

► **Observation 14 (\*)**. *Let  $\mathbf{q} = (q_1, \dots, q_k) \in \mathbb{Z}^k$  be a vector and let  $\alpha \in \mathbb{N}$ . In time  $\mathcal{O}(2^k \cdot 3^\alpha \cdot \alpha \cdot k)$ , one can enumerate all vectors  $\mathbf{p} = (p_1, \dots, p_k) \in \mathbb{Z}^k$  with  $\sum_{i=1}^k |q_i - p_i| \leq \alpha$ .*

By setting  $\alpha = d/c$  and  $q_i = \ell$  for each  $i \in [1, k]$ , Observation 14 implies that we have to compute at most  $2^{|X|} \cdot 3^{d/c} \cdot (|X| + n + d/c + m)^{\mathcal{O}(1)}$  entries of  $D$  to compute  $d^* := \min_{\ell \in [1, m]} \min_{s \in V(X)} D[(|x_1|, \dots, |x_k|), \ell, s]$ . Since each entry can be computed in time  $2^{|X|} \cdot (|X| + n + d/c + m)^{\mathcal{O}(1)}$ , we obtain the stated running time. If  $d^* \leq d$ , the corresponding time series can be found via traceback in the same running time. This shows Theorem 12.

**An FPT-algorithm for the Distance Bound.** In this section, we now obtain an FPT algorithm for the parameter  $d/c$ , removing the running time dependence on  $|X|$ .

► **Theorem 15**. *MSM-MEDIAN can be solved in time  $2^{\mathcal{O}(d/c)} \cdot |I|^{\mathcal{O}(1)}$ . Moreover, when given a yes-instance of MSM-MEDIAN, one can find a median in the same running time.*

For a time series  $x$ , we define  $X_{\text{Close}}(x) := \{y \in X \mid |y| = |x| \text{ and } d_{\text{MSM}(c)}(x, y) \leq 3 \cdot c/2\}$ .

► **Lemma 16**. *Let  $X$  be a set of time series, let  $x \in X$ , and let  $x^*$  be any time series with  $d_{\text{MSM}(c)}(x, x^*) < c/2$ , then*

- $|x^*| = |x|$ ,
- for each  $y \in X_{\text{Close}}(x)$ ,  $d_{\text{MSM}(c)}(y, x^*) = d_{\text{Move}}(y, x^*)$ , and
- for each  $z \in X \setminus X_{\text{Close}}(x)$ ,  $d_{\text{MSM}(c)}(z, x^*) \geq c$ .

**Proof.** First, note that since  $d_{\text{MSM}(c)}(x, x^*) < c/2$ , each optimal transformation forest between  $x$  and  $x^*$  uses only move edges. Hence,  $|x^*| = |x|$ .

Next, we show that for each  $y \in X_{\text{Close}}(x)$ ,  $d_{\text{MSM}(c)}(y, x^*) = d_{\text{Move}}(y, x^*)$ . By the triangle inequality,  $d_{\text{MSM}(c)}(y, x^*) \leq d_{\text{MSM}(c)}(y, x) + d_{\text{MSM}(c)}(x, x^*) < 3 \cdot c/2 + c/2 = 2c$ . Hence, since  $|y| = |x| = |x^*|$ , each transformation forest between  $y$  and  $x^*$  contains the same number of split and merge operations. Since  $d_{\text{MSM}(c)}(y, x^*) < 2c$ , each optimal transformation forest between  $y$  and  $x^*$  uses only move edges which implies that  $d_{\text{MSM}(c)}(y, x^*) = d_{\text{Move}}(y, x^*)$ .

Finally, we show that for each  $z \in X \setminus X_{\text{Close}}(x)$ ,  $d_{\text{MSM}(c)}(z, x^*) \geq c$ . Let  $z \in X \setminus X_{\text{Close}}(x)$ . If  $|z| \neq |x|$ , then since  $|x| = |x^*|$ , each transformation forest between  $z$  and  $x^*$  contains at least one split or merge operation. Consequently,  $d_{\text{MSM}(c)}(z, x^*) \geq c$ . Otherwise, that is, if  $|z| = |x|$ , then  $3 \cdot c/2 < d_{\text{MSM}(c)}(z, x)$  since  $z \notin X_{\text{Close}}(x)$ . By the triangle inequality,  $d_{\text{MSM}(c)}(z, x) \leq d_{\text{MSM}(c)}(z, x^*) + d_{\text{MSM}(c)}(x, x^*) < d_{\text{MSM}(c)}(z, x^*) + c/2$ . Hence,  $3 \cdot c/2 < d_{\text{MSM}(c)}(z, x^*) + c/2$  which implies  $d_{\text{MSM}(c)}(z, x^*) > c$ . ◀

**Proof of Theorem 15.** Let  $I := (X', d)$  be an instance of MSM-MEDIAN where each time series of  $X'$  has length at most  $n$  and let  $(X, \omega)$  be the weighted equivalent of  $X'$ . There is a median that only uses values of  $V(X)$  [6, Lemma 10]. We describe how to find in the stated running time a time series  $x^*$  with this property that minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*)$  or correctly detect that no such time series exists with  $D_{\text{MSM}(c)}^\omega(X, x^*) \leq d$ . We distinguish two cases.

**Case 1:**  $d/c \geq |X|/2$ . Hence,  $|X| \leq 2 \cdot d/c$ . By Lemma 11, the sought time series  $x^*$  has length at most  $n \cdot |X|$ . Due to Theorem 12, we can find a time series with the desired properties that has length at most  $n \cdot |X|$  in time  $4^{|X|} \cdot 3^{d/c} \cdot (|X| + n + d/c)^{\mathcal{O}(1)} \leq 4^{2 \cdot d/c} \cdot 3^{d/c} \cdot (|X| + n + d/c)^{\mathcal{O}(1)} = 48^{d/c} \cdot (|X| + n + d/c)^{\mathcal{O}(1)}$  or detect that no such time series exists with  $D_{\text{MSM}(c)}^\omega(X, x^*) \leq d$ . Hence, if such a time series  $x^*$  is found, we can correctly output that  $I$  is a yes-instance of MSM-MEDIAN and return the found time series. Otherwise, by the above, we can correctly output that  $I$  is a no-instance of MSM-MEDIAN.

**Case 2:**  $d/c < |X|/2$ . The idea is as follows: If  $I$  is a yes-instance of MSM-MEDIAN, then there is a median  $x^*$  with  $D_{\text{MSM}(c)}^\omega(X, x^*) \leq d$  and some time series  $\tilde{x} \in X$  with  $d_{\text{MSM}(c)}(\tilde{x}, x^*) \leq d/|X| < c/2$ . Lemma 16 now implies: for each time series  $y \in X_{\text{Close}}(\tilde{x})$ ,  $d_{\text{MSM}(c)}(y, x^*) = d_{\text{Move}}(y, x^*)$  and for each time series  $z \in X \setminus X_{\text{Close}}(\tilde{x})$ ,  $d_{\text{MSM}(c)}(z, x^*) \geq c$ . This implies that  $Z := X \setminus X_{\text{Close}}(\tilde{x})$  contains at most  $d/c$  time series. Hence, to find a median, the main algorithmic difficulty lies in finding a time series of length  $|\tilde{x}|$  that uses only points of  $V(X)$  and minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*) = D_{\text{MSM}(c)}^\omega(Z, x^*) + \sum_{y \in X_{\text{Close}}(\tilde{x})} (\omega(y) \cdot d_{\text{Move}}(y, x^*))$ . We show that this can be done in time  $4^{|\tilde{x}|} \cdot 3^{d/c} \cdot (|X| + n + d/c)^{\mathcal{O}(1)}$  by modifying the DP of Theorem 12. Essentially the idea is that since for each time series  $y \in X_{\text{Close}}(\tilde{x})$ ,  $d_{\text{MSM}(c)}(y, x^*) = d_{\text{Move}}(y, x^*)$ , the transformation forest between  $y$  and  $x^*$  is fixed and we do not need to store current positions of time series in  $X_{\text{Close}}(\tilde{x})$  as dimensions in the DP-table.

▷ **Claim 17 (\*)**. Let  $Z$  be a set of time series of length at most  $n_Z$  each, let  $Y$  be a non-empty set of time series of length  $n_Y$  each, let  $\omega : Z \cup Y \rightarrow \mathbb{N}^+$ , and let  $d \in \mathbb{R}$ . In time  $4^{|Z|} \cdot 3^{d/c} \cdot (|Z \cup Y| + \max(n_Z, n_Y) + d/c)^{\mathcal{O}(1)}$  one can find a time series  $x^*$  that

- contains only points of  $V(Z) \cup V(Y)$ , has length  $n_Y$ , and
- minimizes  $d_{Z,Y}(x^*) := D_{\text{MSM}(c)}^\omega(Z, x^*) + \sum_{y \in Y} (\omega(y) \cdot d_{\text{Move}}(y, x^*))$

or correctly output that there is no such time series  $x^*$  with  $d_{Z,Y}(x^*) \leq d$ .

The algorithm for  $d/c < |X|/2$  now works as follows: Branch into all possibilities for  $\tilde{x}$ . That is, iterate over all time series  $x \in X$  and compute the sets  $Y := X_{\text{Close}}(x)$  and  $Z := X \setminus Y$ . If  $|Z| > d/c$ , then continue with the next time series since  $x$  is no candidate for  $\tilde{x}$ . Otherwise, apply the algorithm behind Claim 17 for the sets  $Z$  and  $Y$ . If this algorithm returns that there is no time series  $x^*$  with the desired property, then  $x$  is not a candidate for  $\tilde{x}$  or  $I$  is a no-instance of MSM-MEDIAN. Otherwise, store the time series  $x^*$  that minimizes  $D_{\text{MSM}(c)}^\omega(Z, x^*) + \sum_{y \in Y} (\omega(y) \cdot d_{\text{MSM}(c)}(y, x^*)) \leq d$ . After iterating over all time series of  $X$ , output the stored time series  $x^*$  that minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*)$ . If no such time series was found, output that  $I$  is a no-instance of MSM-MEDIAN.

By the above, this algorithm is correct. It remains to show the running time. Since for any two time series  $x$  and  $y$ ,  $d_{\text{MSM}(c)}(x, y)$  can be computed in  $\mathcal{O}(|x| \cdot |y|)$  time [12], each individual step of this algorithm runs in  $|I|^{\mathcal{O}(1)}$  time. For each time series  $x$  with  $|X \setminus X_{\text{Close}}(x)| \leq d/c$ , the algorithm behind Claim 17 can be applied in time  $4^{|X \setminus X_{\text{Close}}(x)|} \cdot 3^{d/c} \cdot |I|^{\mathcal{O}(1)} \leq 12^{d/c} \cdot |I|^{\mathcal{O}(1)}$ . Consequently, this algorithm runs in time  $12^{d/c} \cdot |I|^{\mathcal{O}(1)}$ .

Since in both cases the running time is at most  $48^{d/c} \cdot |I|^{\mathcal{O}(1)}$ , the statement holds. ◀

Finally, let us observe that the concrete value of  $d$  need not be known in advance.

► **Corollary 18.** *Let  $X$  be a sequence of time series of length at most  $n$  each, then we can find in time  $2^{\mathcal{O}(d/c)} \cdot (|X| + n + d/c)^{\mathcal{O}(1)}$  a time series  $x^*$  that minimizes  $D_{\text{MSM}(c)}^\omega(X, x^*) = d$ .*

---

**References**

---

- 1 Saeed Reza Aghabozorgi, Ali Seyed Shirkhorshidi, and Ying Wah Teh. Time-series clustering - A decade review. *Inf. Syst.*, 53:16–38, 2015. doi:10.1016/j.is.2015.04.007.
- 2 Markus Brill, Till Fluschnik, Vincent Froese, Brijnesh J. Jain, Rolf Niedermeier, and David Schultz. Exact mean computation in dynamic time warping spaces. *Data Min. Knowl. Discov.*, 33(1):252–291, 2019. doi:10.1007/s10618-018-0604-8.
- 3 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15)*, pages 79–97. IEEE Computer Society, 2015.
- 4 Laurent Bulteau, Vincent Froese, and Rolf Niedermeier. Tight hardness results for consensus problems on circular strings and time series. *SIAM J. Discret. Math.*, 34(3):1854–1883, 2020. doi:10.1137/19M1255781.
- 5 Jana Holznigenkemper, Christian Komusiewicz, and Bernhard Seeger. Exact and heuristic approaches to speeding up the MSM time series distance computation. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM '23)*, pages 451–459. SIAM, 2023. doi:10.1137/1.9781611977653.ch51.
- 6 Jana Holznigenkemper, Christian Komusiewicz, and Bernhard Seeger. On computing exact means of time series using the move-split-merge metric. *Data Min. Knowl. Discov.*, 37(2):595–626, 2023.
- 7 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.
- 8 Weiwei Jiang. Time series classification: Nearest neighbor versus deep learning models. *SN Appl. Sci.*, 2(4):721, 2020. doi:10.1007/s42452-020-2506-9.
- 9 Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Min. Knowl. Discov.*, 29:565–592, 2015. doi:10.1007/s10618-014-0361-2.
- 10 John Paparrizos and Luis Gravano. Fast and accurate time-series clustering. *ACM Trans. Database Syst.*, 42(2):8:1–8:49, 2017. doi:10.1145/3044711.
- 11 John Paparrizos, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. Debunking four long-standing misconceptions of time-series distance measures. In *Proceedings of the 2020 International Conference on Management of Data (SIGMOD '20)*, pages 1887–1905. ACM, 2020. doi:10.1145/3318464.3389760.
- 12 Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. *IEEE Trans. Knowl. Data Eng.*, 25(6):1425–1438, 2013. doi:10.1109/TKDE.2012.88.



# Fixed-Parameter Algorithms for Fair Hitting Set Problems

Tanmay Inamdar ✉ 

University of Bergen, Norway

Lawqueen Kanesh ✉

Indian Institute of Technology, Jodhpur, India

Madhumita Kundu ✉

University of Bergen, Norway

Nidhi Purohit ✉

University of Bergen, Norway

Saket Saurabh ✉

Institute of Mathematical Sciences, Chennai, India

University of Bergen, Norway

---

## Abstract

Selection of a group of representatives satisfying certain fairness constraints, is a commonly occurring scenario. Motivated by this, we initiate a systematic algorithmic study of a *fair* version of HITTING SET. In the classical HITTING SET problem, the input is a universe  $\mathcal{U}$ , a family  $\mathcal{F}$  of subsets of  $\mathcal{U}$ , and a non-negative integer  $k$ . The goal is to determine whether there exists a subset  $S \subseteq \mathcal{U}$  of size  $k$  that *hits* (i.e., intersects) every set in  $\mathcal{F}$ . Inspired by several recent works, we formulate a fair version of this problem, as follows. The input additionally contains a family  $\mathcal{B}$  of subsets of  $\mathcal{U}$ , where each subset in  $\mathcal{B}$  can be thought of as the group of elements of the same *type*. We want to find a set  $S \subseteq \mathcal{U}$  of size  $k$  that (i) hits all sets of  $\mathcal{F}$ , and (ii) does not contain *too many* elements of each type. We call this problem FAIR HITTING SET, and chart out its tractability boundary from both classical as well as multivariate perspective. Our results use a multitude of techniques from parameterized complexity including classical to advanced tools, such as, methods of representative sets for matroids, FO model checking, and a generalization of best known kernels for HITTING SET.

**2012 ACM Subject Classification** Theory of computation → Fixed parameter tractability

**Keywords and phrases** Fairness, Parameterized Algorithms, Hitting Set

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.55

**Related Version** *Full Version*: <https://arxiv.org/abs/2307.08854>

**Funding** *Tanmay Inamdar*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).

*Madhumita Kundu*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).

*Nidhi Purohit*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416).

*Saket Saurabh*: Supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).

## 1 Introduction

Imagine a scenario of selecting a committee of size  $k$  from a group of people  $\mathcal{U}$ . We need a committee of people with some given attributes. These kinds of “attribute hitting” scenarios is modeled by a family  $\mathcal{F}$  over  $\mathcal{U}$ , where for each attribute  $\mathcal{A}$ , we have a set  $\mathcal{F}$  containing



© Tanmay Inamdar, Lawqueen Kanesh, Madhumita Kundu, Nidhi Purohit, and Saket Saurabh; licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 55; pp. 55:1–55:14

Leibniz International Proceedings in Informatics



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

people in  $\mathcal{U}$  who have the attribute  $\mathcal{A}$ . As is life, not always every set of people can work collectively. In particular, the committee cannot operate smoothly if we select more than the desired number of people from a set  $B \subseteq \mathcal{U}$ . These conflicts are modeled by another family,  $\mathcal{B}$  over  $\mathcal{U}$ , and a function  $f : \mathcal{B} \rightarrow \mathbb{N}$ , which says that  $f(B)$  is the maximum number of people from a set  $B \in \mathcal{B}$  that can serve on the committee. Specifically, we want a committee that is a hitting set for attributes and has a set of people who are “conflict free”. This paper aims to undertake a systematic study of a generalization of Hitting Set, which models such scenarios, and study this problem in the realm of parameterized complexity.

Indeed, HITTING SET is one of the 21 problems proven to be NP-complete by [6]. Recall, in this problem, we are given a set system  $(\mathcal{U}, \mathcal{F})$ , and an integer  $k$ . Here,  $\mathcal{U}$  is a finite set of elements known as *universe* and  $\mathcal{F}$  is a family of subsets of  $\mathcal{U}$ . The objective is to determine whether there exists a subset  $S \subseteq \mathcal{U}$  such that  $S$  *hits* all sets in  $\mathcal{F}$ , i.e., for every  $F_i \in \mathcal{F}$ ,  $S \cap F_i \neq \emptyset$ . HITTING SET is closely related to the SET COVER problem. These two problems, along with a particularly interesting special case thereof, namely that of VERTEX COVER, are some of the most extensively studied problems in the field of approximation algorithms and parameterized complexity. HITTING SET problem is of a particular interest, because many combinatorial problems can be modeled as instances of HITTING SET.

Motivated from real-life applications, there has been a growing interest on the fairness aspect of various problems and algorithms developed. This has led to the whole new field of algorithmic fairness. Depending on the specific application, there are numerous ways to define the notion of *fairness*. One of the earliest definitions of fairness comes from [8], who defined fair versions of edge deletion problems. This was motivated from the following scenario. Suppose the graph models a communication network, with each edge being a link between a pair of nodes. In order to achieve acyclicity in the network, some links need to be disconnected. However, from the perspective of each node, it is desirable that fewest possible links incident to it are disconnected. Thus, we wish to disconnect links in a fair or equitable manner for the nodes.

Subsequently, this notion was extended by [11, 7] to define fair versions of vertex deletion problems. In this model, we want to delete a subset of vertices in order to achieve a certain graph property, such that each vertex has fewest possible neighbors deleted. As a concrete example, in a fair version of VERTEX COVER in this model, we want to find a vertex cover  $S$ , such that each vertex outside  $S$  has fewest neighbors in  $S$ . Recently, [1] studied a generalization of this, called SPARSE HITTING SET. The input to SPARSE HITTING SET consists of  $(\mathcal{U}, \mathcal{F}, \mathcal{B})$ , where  $\mathcal{U}$  is the universe, and  $\mathcal{F}$  and  $\mathcal{B}$  are two families of subsets of  $\mathcal{U}$ . The goal is to find a hitting set  $S \subseteq \mathcal{U}$  for  $\mathcal{F}$  such that  $k := \max_{B_i \in \mathcal{B}} |B_i \cap S|$  is minimized. Here,  $k$  is called the *sparseness* of the solution. Note that SPARSE HITTING SET generalizes FAIR VERTEX COVER as defined above. Along a similar line, [5] considered *conflict-free* versions of various problems, including HITTING SET. In CONFLICT FREE  $d$ -HITTING SET, we are given an instance  $(\mathcal{U}, \mathcal{F}, k)$  of HITTING SET, and a conflict graph  $H = (\mathcal{U}, E)$ , and the goal is to find a hitting set  $S \subseteq \mathcal{U}$  of size at most  $k$ , such that  $S$  induces an independent set in the conflict graph  $H$ .

## Our Problem

Along the same line of work, we define a fair version of HITTING SET, which captures all of the aforementioned problems, and much more. Formally, the problem is defined as follows.

## FAIR HITTING SET

**Input.** An instance  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$ , where  $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$  is the universe;  $\mathcal{B}$  and  $\mathcal{F}$  are two families of subsets of  $\mathcal{U}$ , where  $\mathcal{F} = \{F_1, F_2, \dots, F_m\}$ , and  $\mathcal{B} = \{B_1, B_2, \dots, B_\ell\}$ , and  $k$  is a positive integer.

**Task.** Determine whether there exists  $S \subseteq \mathcal{U}$ , with the following properties.

- $|S| \leq k$ ,
- $S$  is a *hitting set* for  $\mathcal{F}$ , i.e., for every  $F_i \in \mathcal{F}$ ,  $S \cap F_i \neq \emptyset$ , and
- For every  $B_j \in \mathcal{B}$ ,  $|S \cap B_j| \leq f(B_j)$ .

We refer to a set  $S \subseteq \mathcal{U}$  satisfying the above properties as a *fair hitting set* for  $\mathcal{F}$ , and use  $|\mathcal{I}|$  to denote the size of the instance  $\mathcal{I}$ .

We note that FAIR HITTING SET generalizes SPARSE HITTING SET. Given an instance  $(\mathcal{U}, \mathcal{F}, \mathcal{B})$  of SPARSE HITTING SET, we iteratively solve instances  $\mathcal{I}_i$  of FAIR HITTING SET for  $i = 1, 2, \dots$ . Here, an instance  $\mathcal{I}_i$  of FAIR HITTING SET is given by  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f_i, |\mathcal{U}|)$ , where  $f_i(B_j) = i$  for all  $B_j \in \mathcal{B}$ . For the smallest  $i$  such that  $\mathcal{I}_i$  is a yes-instance of FAIR HITTING SET, we stop and conclude that  $i$  is the optimal sparseness of the given instance of SPARSE HITTING SET. We note that FAIR HITTING SET also generalizes the setting considered by [5].

## 1.1 Our Results, Techniques, and Relation to Hitting Set

First, we observe that FAIR HITTING SET is a generalization of HITTING SET, by setting  $\mathcal{B} = \emptyset$ . Thus, FAIR HITTING SET inherits all lower bound results from HITTING SET, namely, in general the problem is NP-hard as well as W[2]-hard parameterized by  $k$ , the solution size [2]. However, note that in the hard instances of HITTING SET, the sets in  $\mathcal{F}$  can intersect arbitrarily. Indeed, consider an extreme case, when the sets in  $\mathcal{F}$  are pairwise disjoint. In this setting HITTING SET is trivial to solve – an optimal solution must contain exactly one element from each set of  $\mathcal{F}$ . In contrast, we show that FAIR HITTING SET remains NP-hard, as well as W[1]-hard w.r.t.  $k$  – and thus unlikely to be FPT – even in this simple setting. In particular, we show the following lower bound results, which are proved formally in the full version.

► **Theorem 1.** FAIR HITTING SET remains NP-hard when (1) the sets in  $\mathcal{F}$  are pairwise disjoint, (2) each element appears in at most two distinct  $B_i$ 's in  $\mathcal{B}$ , and (3) each  $B_i \in \mathcal{B}$  has size exactly 2. Furthermore, assuming ETH, it is not possible to solve FAIR HITTING SET in time  $2^{o(t)}$ , where  $t = \max\{|\mathcal{U}|, |\mathcal{F}|, |\mathcal{B}|\}$ .

FAIR HITTING SET is W[1]-hard when parameterized by  $k$ , even when the sets in  $\mathcal{F}$  are pairwise disjoint, and each  $B_i \in \mathcal{B}$  has size exactly 2.

The first result is obtained via a reduction from a problem of finding a “rainbow matching” on a path, and for the second result we give a parameter preserving reduction from  $k$ -MULTICOLORED INDEPENDENT SET. Given these lower bound results (Theorem 1), we study FAIR HITTING SET under specific assumptions on the instance  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f, k)$ . A natural question is: *under which assumptions?* To answer this we look at the known fixed-parameter tractability results for HITTING SET.

### Hitting Set in Parameterized Complexity

HITTING SET is known to be  $W[2]$ -complete parameterized by the solution size in general. In other words, under widely believed complexity theoretic assumptions, it does not admit an FPT algorithm parameterized by the solution size. This motivates the study of HITTING SET in special cases. One particularly interesting case is VERTEX COVER when the size of each set in  $\mathcal{F}$  is exactly two. VERTEX COVER is the most extensively studied problem in the parameterized complexity with a number of results in the FPT algorithms and kernelization in general graphs as well as special classes of graphs. Many of the techniques and results developed for VERTEX COVER also extend  $d$ -HITTING SET, where each set in  $\mathcal{F}$  has size at most  $d$ , for some constant  $d$ . More generally, HITTING SET is known to be FPT and admits a polynomial kernel in the case when the incidence graph  $G_{\mathcal{U},\mathcal{F}}$ , which is the bipartite graph on the vertex set  $\mathcal{U} \uplus \mathcal{F}$  with edges denoting the set-containment, is  $K_{i,j}$ -free. That is, no  $i$  sets in  $\mathcal{F}$  contain  $j$  elements in common, where  $i$  and  $j$  are assumed to be constants. This setting generalizes all the above settings as well as when the  $G_{\mathcal{U},\mathcal{F}}$  is  $d$ -degenerate (since such graphs are  $K_{d+1,d+1}$ -free).

### Our Algorithmic Results

Notably, we are able to extend almost all of the fixed-parameter tractability results for HITTING SET mentioned in the previous paragraph, under suitable assumptions on the set system  $(\mathcal{U}, \mathcal{B})$ . We give a summary of our results in Figure 1.

More specifically, we obtain our results in the following steps. Consider a special case FAIR HITTING SET, when the sets in  $\mathcal{F}$  are pairwise disjoint, and each element appears in at most  $q$  sets in  $\mathcal{B}$ . Note that the first part of Theorem 1 implies that the problem is NP-hard even when  $q = 2$ . On the other hand, when  $q = 1$ , i.e., when both  $\mathcal{F}$  and  $\mathcal{B}$  are families of pairwise disjoint sets, then we observe FAIR HITTING SET can be solved in polynomial time. Thus,  $q = 1$  to  $2$  is a sharp transition between the tractability of the problem. Although the problem is NP-hard even for constant values of  $q$ , the following results are interesting in this setting. In particular, we show that the problem is FPT, and admits a polynomial kernel parameterized by  $k$ , if  $q$  is a constant.

► **Theorem 2.** *Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. Then, FAIR HITTING SET can be solved in time  $2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$  time, when every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and any pair of sets in  $\mathcal{F}$  are pairwise disjoint. Further, FAIR HITTING SET admits a kernel of size  $\mathcal{O}(kq^2 \binom{kq}{q} \log k)$ .*

Next we generalize Theorem 2 to a scenario where every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and at most  $d$  sets in  $\mathcal{F}$ .

► **Theorem 3.** *Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. Then, FAIR HITTING SET can be solved in time  $k^{\mathcal{O}(dk)} 2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$  time, when every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and at most  $d$  sets in  $\mathcal{F}$ .*

These results, Theorems 2 and 3, are obtained by the key observation that the problem can be modeled as finding a hitting set for  $\mathcal{F}$  that is also an independent set in a suitably defined partition matroid that encodes the constraints imposed by  $(\mathcal{U}, \mathcal{B}, f)$ . This enables us to use the representative sets toolkit developed for matroids. This result is discussed in Section 3.

Next we consider a generalization of the above setting, where (1) each element appears in at most  $q$  sets in  $\mathcal{B}$ , and (2) the  $G_{\mathcal{U},\mathcal{F}}$  is  $K_{d,d}$ -free. In this case, we combine the techniques developed in HITTING SET literature in the  $K_{d,d}$ -free setting, as well as, the representative sets based techniques developed in Section 3, to obtain the following result.

► **Theorem 4.** *Given an instance  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f, k)$  of FAIR HITTING SET, such that  $G_{\mathcal{U}, \mathcal{F}}$  is  $K_{d,d}$ -free, and the frequency of each element in  $\mathcal{B}$  is bounded by  $q$ , one can find an equivalent instance  $\mathcal{I}' = (\mathcal{U}', \mathcal{F}', \mathcal{B}', f', k')$  of FAIR HITTING SET in polynomial time, such that  $|\mathcal{U}'| = \mathcal{O}(k^{d^2+q}d^dq^q)$ ,  $|\mathcal{F}'| \leq dk^d$ , and  $|\mathcal{B}'| = \mathcal{O}(k^{d^2+q} \cdot d^dq^{q+1})$ , where  $d$  and  $q$  are assumed to be constants.*

Finally, we reach our most general case, where suppose (1) the  $(\mathcal{U}, \mathcal{B})$  incidence graph is “nowhere dense” (defined formally in the full version; this class includes planar, excluded minor, bounded degree, and bounded expansion graphs), and (2) the  $(\mathcal{U}, \mathcal{F})$  incidence graph is  $K_{d,d}$ -free. In this case, we obtain an FPT algorithm, parameterized by  $k$  and  $d$ . This result is in two steps. First, we proceed as prior to the case when each element appears in  $f(k, d)$  sets of  $\mathcal{F}$  (cf. Theorem 4). Next, since  $(\mathcal{U}, \mathcal{B})$  incidence graph is nowhere dense, we reduce the problem of finding a FAIR HITTING SET to FO model checking procedure on nowhere dense graphs, which is known to be FPT in the size of the formula. In particular, we show that the problem can be encoded by a variant of INDUCED SUBGRAPH ISOMORPHISM on nowhere dense graphs, where the size of the host graph we are searching for can be bounded by a function of  $k$ ,  $d$  and the graph class.

► **Theorem 5.** *Let  $\mathcal{G}$  be a nowhere dense graph class. Let  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f, k)$  be an instance of FAIR HITTING SET such that the incidence graph  $G := G_{\mathcal{U}, \mathcal{B}} \in \mathcal{G}$ , and  $G_{\mathcal{U}, \mathcal{F}}$  is  $K_{d,d}$ -free for some  $d \geq 1$ . Then, one can solve FAIR HITTING SET on  $\mathcal{I}$  in time  $h(k, d) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$ , for some function  $h(\cdot, \cdot)$ .*

## 2 Preliminaries

For an integer  $\ell \geq 1$ , we use the notation  $[\ell] := \{1, 2, \dots, \ell\}$ . Let  $\mathcal{R} = (\mathcal{U}, \mathcal{S})$  be a *set system*, where  $\mathcal{U}$  is a finite set of elements (also called the *ground set* or the *universe*), and  $\mathcal{S}$  is a family of subsets of  $\mathcal{U}$ . For an element  $u \in \mathcal{U}$ , and any  $\mathcal{S}' \subseteq \mathcal{S}$ , we use the notation  $\mathcal{S}'(u) := \{S \in \mathcal{S}' : u \in S\}$ , i.e.,  $\mathcal{S}'(u)$  is the sub-family of sets from  $\mathcal{S}'$  that contain  $u$ . For a subset  $R \subseteq \mathcal{U}$ , we denote  $\mathcal{S} - R := \{S \setminus R : S \in \mathcal{S}\}$ . We use  $G_{\mathcal{U}, \mathcal{S}}$  to denote the incidence graph corresponding to the set system  $(\mathcal{U}, \mathcal{S})$ , i.e.,  $G_{\mathcal{U}, \mathcal{S}}$  is a bipartite graph with bipartition  $\mathcal{U} \uplus \mathcal{S}$ , such that there is an edge between an element  $e \in \mathcal{U}$  and a set  $S \in \mathcal{S}$  iff  $e \in S$ .

In this paper, we work with finite, simple, undirected graphs. We use the standard graph theoretic notation and terminology, as defined in [3].

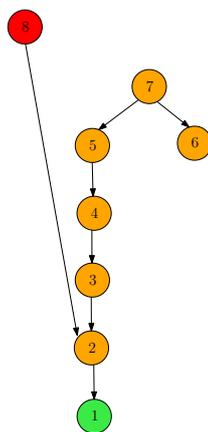
## 3 FPT Algorithm and Kernel Based on Representative Sets

In this section we design an algorithm and a kernel for a special case of FAIR HITTING SET, using methods based on representative sets [4, 10]. Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. The first special case we consider is the following: every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and any pair of sets in  $\mathcal{F}$  are pairwise disjoint.

Before this, however, we consider the special case of  $q = 1$ , i.e., when any pair of sets in  $\mathcal{B}$ , as well as that in  $\mathcal{F}$  are disjoint. In this case, we can solve the problem in polynomial time, by reducing it to the problem of finding maximum flow in an auxiliary directed graph, defined as follows. The vertices of the graph are  $\mathcal{B} \uplus \mathcal{U} \uplus \mathcal{F} \uplus \{s, t\}$ . First, we add arcs (i.e., directed edges) from source  $s$  to each  $B_j \in \mathcal{B}$ , with capacity  $f(B_j)$ . Next, for every  $u \in B_j$ , we add an arc  $(B_j, u)$  of capacity 1. Similarly, for each  $u \in F_i$ , we add an arc  $(u, F_i)$ , of capacity 1. Finally, we add arcs  $(F_i, t)$  of capacity  $\infty$ . It is straightforward to show that there exists a flow of value  $k$  in the graph iff there exists a fair hitting set of size  $k$ . We omit the details.

No.	$\mathcal{G}_{\mathcal{U},\mathcal{B}}$	$\mathcal{G}_{\mathcal{U},\mathcal{F}}$	Results
1	$q = 1$ ( $B_i$ 's are disjoint)	$d = 1$ ( $F_i$ 's are disjoint)	Polynomial time
2	$q = 2$	$d = 1$	NP-Hard No sub-exp algo(full version)
3	$q$	$d = 1$	$2^{\mathcal{O}(qk)} \cdot  \mathcal{I} ^{\mathcal{O}(1)}$ (Theorem 2)  $\mathcal{O}(kq^2 \binom{kq}{q} \log k)$ (kernel) (Theorem 12)
4	$q$	$d$	$k^{\mathcal{O}(dk)} 2^{qk}  \mathcal{I} ^{\mathcal{O}(1)}$ (Theorem 3)
5	$q$	$K_{d,d}$ -free	$k^{\mathcal{O}(d^2+q)} d^d q^{q+1}$ (kernel) (Theorem 4)
6	apex-minor free	$K_{d,d}$ -free	FPT/ $(k + d)$ (full version)
7	nowhere dense	$K_{d,d}$ -free	FPT/ $(k + d)$ (Theorem 5)
8	$K_{2,2}$ -free	$d = 1$	$W[1]$ -Hard/ $k$ (full version)

■ **Figure 1** An overview of different results obtained in this paper. In the second (resp. third) column, we state the assumption on the set system  $(\mathcal{U}, \mathcal{B})$  (resp.  $(\mathcal{U}, \mathcal{F})$ ). In rows 1-5 (resp. rows 1-4)  $q$  (resp.  $d$ ) denotes the maximum frequency of an element in  $\mathcal{B}$  (in  $\mathcal{F}$ ). In the last column, we mention our results in the respective settings, and give corresponding references. Note that some of the references can be found in the full version.



■ **Figure 2** A Hasse diagram of the settings considered in Figure 1, where the number in each node corresponds to the row in the table. An arrow from node  $i$  to node  $j$  indicates that the setting in row  $i$  generalizes the setting in row  $j$ . Nodes colored in green, orange (resp. red) color indicate that the setting is solvable in polynomial, FPT time (resp. is  $W[1]$ -hard).

Note that  $q \geq 2$ , but the sets in  $\mathcal{F}$  are pairwise disjoint, the problem is NP-hard. In this case, To design both our algorithm and the kernel we first embed the fairness constraints imposed by  $\mathcal{B}$  in a combinatorial object called a *partition matroid*. A partition matroid is a set system  $\mathcal{M} = (E, \mathcal{I})$ , defined as follows. The ground set  $E$  is partitioned into  $\ell$  subsets  $E_1 \uplus E_2 \uplus \dots \uplus E_\ell$ , such that a set  $S \subseteq E$  belongs to the family  $\mathcal{I}$  iff for each  $1 \leq j \leq \ell$ , it holds that  $|E_j \cap S| \leq k_j$ , where  $k_1, k_2, \dots, k_\ell$  are non-negative integers.

It might be observed that the definition of a partition matroid closely resembles the fairness constraints, i.e., for each  $B_j$ , the hitting set  $H$  must satisfy  $|H \cap B_j| \leq f(B_j)$ . However, this idea does not quite work, since the sets  $B_j \in \mathcal{B}$  are not disjoint – indeed, otherwise we could solve the problem in polynomial time, as discussed earlier. Nevertheless, we can salvage the situation by making  $q$  distinct copies of every element  $u \in \mathcal{U}$ , and replacing each of the occurrences of  $u$  in  $q$  distinct  $B_j$ 's with a unique copy. The resulting set system is a partition matroid that exactly captures the fairness constraints. Correspondingly, in each set of  $\mathcal{F}$ , we replace an original element with all of its  $q$  copies. Recall that we want to find a hitting set for  $\mathcal{F}$ ; however, in the new formulation, we must now ensure that if we pick *at least* one copy of element in the solution, we pick *all* of its copies in the solution.

Thus, our solution is an independent set of  $\mathcal{M}$  that (1) is a hitting set for  $\mathcal{F}$ , and (2) picks either 0 or  $q$  copies of every element. To find such a solution in time FPT in  $k$  and  $q$  (resp. to reduce the size of the instance), we use a sophisticated tool developed in parameterized complexity, called *representative sets*. Later, we generalize this idea to the case where every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and at most  $d$  sets in  $\mathcal{F}$ . In the next section, we formally define the partition matroid, and in the subsequent sections, we apply the toolkit of representative sets to design our FPT algorithm and the kernel.

### 3.1 Partition Matroid and Our Solution

In a partition matroid we have a universe  $\tilde{U}$ , partitioned into  $\tilde{U}_1, \dots, \tilde{U}_\ell$ , together with positive integers  $k_1, \dots, k_\ell$ , and a family of independent sets  $\mathcal{I}$ , such that  $X \subseteq \tilde{U}$  is in  $\mathcal{I}$  if and only if  $|X \cap \tilde{U}_i| \leq k_i, i \in [\ell]$ .

Let  $(\mathcal{U}, \mathcal{B})$  be the given set system such that each element  $u \in \mathcal{U}$  appears in *at most*  $q$  sets of  $\mathcal{B}$ . For an element  $u \in \mathcal{U}$ , let  $q(u) \leq q$  denote the number of sets in  $\mathcal{B}$ , that  $u$  appears in. Further, for an element  $u \in \mathcal{U}$ , let  $\text{copies}(u) = \{u^1, u^2, \dots, u^{q(u)}\}$ . We define

$$\tilde{U} = \bigcup_{u \in \mathcal{U}} \text{copies}(u).$$

Next, we need to define a partition of  $\tilde{U}$ . Towards this, we use the information about the sets in  $\mathcal{B}$ . We know that each element  $u \in \mathcal{B}$  appears in  $q(u)$  sets and we have made  $q(u)$  copies of  $u$ , thus we use *distinct* and *unique* copy of  $u$  in each sets in  $\mathcal{B}$  in which  $u$  appears. This results in  $\tilde{\mathcal{B}} = \{\tilde{B}_i : B_i \in \mathcal{B}\}$ , where  $\tilde{B}_i$  is the set corresponding to an original set  $B_i$ , after replacing elements with their copies. Observe that for every pair of indices  $i \neq j$  we have that  $\tilde{B}_i \cap \tilde{B}_j = \emptyset$  and  $\cup_i \tilde{B}_i = \tilde{U}$ . This immediately gives a partition of  $\tilde{U}$ . Finally, we define  $k_i = f(B_i)$ . This completes the description of the partition matroid we will be using. We will call this matroid as  $\mathcal{M} = (\tilde{U}, \mathcal{I})$

Given a subset  $X \subseteq \tilde{U}$ , we define a set associated with  $X$ , called  $\text{projection}(X)$  as follows. The set  $\text{projection}(X) \subseteq \mathcal{U}$ , contains an element  $u \in \mathcal{U}$  if and only if  $\text{copies}(u) \cap X \neq \emptyset$ . Similarly, we define a notion of embedding. For a set  $A \subseteq \mathcal{U}$ , let  $\text{embed}(A) = \cup_{u \in A} \text{copies}(u)$ . This brings us to the following lemma which relates our problem and finding an independent set in the matroid.

► **Lemma 6.** *An input  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  is a yes-instance if and only if there exists an independent set  $X \in \mathcal{I}$  of the matroid  $\mathcal{M} = (\tilde{\mathcal{U}}, \mathcal{I})$  such that (1)  $|\text{projection}(X)| \leq k$ , (2)  $X = \text{embed}(\text{projection}(X))$ , and (3)  $\text{projection}(X)$  is a hitting set for  $\mathcal{F}$ .*

**Proof.** In the forward direction, let  $S \subseteq \mathcal{U}$  be a solution to the original problem. Suppose  $S = \{s_1, s_2, \dots, s_k\}$ . Consider  $X = \text{embed}(S)$ . Observe that by definition,  $S = \text{projection}(X)$ . So  $|\text{projection}(X)| = k$  and  $\text{projection}(X)$  is a hitting set for  $\mathcal{F}$ . We claim that  $X$  is an independent set because if not, there exists a part  $\tilde{B}_i$  which satisfies that  $|X \cap \tilde{B}_i| > k_i = f(B_i)$ . As  $S = \text{projection}(X)$  and for every  $u \in \tilde{B}_i$ ,  $B_i$  contains a  $v$  such that  $u \in \text{copies}(v)$ , this results in  $|S \cap B_i| > k_i$  implying  $|S \cap B_i| > f(B_i)$  which contradicts that  $S$  is a valid solution to the original problem.

In the reverse direction, let  $X \in \mathcal{I}$  be an independent set satisfying both the conditions. Let  $S = \text{projection}(X)$ . We claim that,  $S$  is a solution for the original instance because if not, there exist a part  $B_i$  which satisfies that  $|S \cap B_i| > f(B_i) = k_i$ . As  $X = \text{embed}(S)$  and for every  $u \in B_i$ , a unique copy from  $\text{copies}(u)$  is contained in  $\tilde{B}_i$ , which results in  $|X \cap \tilde{B}_i| > k_i$  which contradicts that  $X$  is an independent set. ◀

### 3.2 Computation of the Desired Independent Set

In this section we give an algorithm to compute an independent set  $X \in \mathcal{I}$  of the matroid  $\mathcal{M} = (\tilde{\mathcal{U}}, \mathcal{I})$  such that  $|\text{projection}(X)| \leq k$  and  $\text{projection}(X)$  is a hitting set for  $\mathcal{F}$  (as given by Lemma 6). We will design a dynamic programming algorithm based on representative families to compute the desired independent set. Towards this we first give the required definitions. We start with the definition of an  $\ell$ -representative family.

► **Definition 7** ( $\ell$ -Representative Family). *Given a matroid  $M = (E, \mathcal{I})$  and a family  $\mathcal{S}$  of subsets of  $E$ , we say that a subfamily  $\hat{\mathcal{S}} \subseteq \mathcal{S}$  is  $\ell$ -representative for  $\mathcal{S}$  if the following holds: for every set  $Y \subseteq E$  of size at most  $\ell$ , if there is a set  $X \in \mathcal{S}$  disjoint from  $Y$  with  $X \cup Y \in \mathcal{I}$ , then there is a set  $\hat{X} \in \hat{\mathcal{S}}$  disjoint from  $Y$  with  $\hat{X} \cup Y \in \mathcal{I}$ . If  $\hat{\mathcal{S}} \subseteq \mathcal{S}$  is  $\ell$ -representative for  $\mathcal{S}$  we write  $\hat{\mathcal{S}} \subseteq_{rep}^{\ell} \mathcal{S}$ .*

In other words if some independent set in  $\mathcal{S}$  can be extended to a larger independent set by adding  $\ell$  new elements, then there is a set in  $\hat{\mathcal{S}}$  that can be extended by the same  $\ell$  elements. We say that a family  $\mathcal{S} = \{S_1, \dots, S_t\}$  of sets is a  $p$ -family if each set in  $\mathcal{S}$  is of size  $p$ .

► **Proposition 8** ([4, 9, Theorem 3.8, Theorem 1.3]). *Let  $M = (E, \mathcal{I})$  be a partition matroid,  $\mathcal{S} = \{S_1, \dots, S_t\}$  be a  $p$ -family of independent sets. Then there exists  $\hat{\mathcal{S}} \subseteq_{rep}^{\ell} \mathcal{S}$  of size  $\binom{p+\ell}{p}$ . Furthermore, given a representation  $A_M$  of  $M$  over a field  $\mathbb{F}$ , there is a deterministic algorithm computing  $\hat{\mathcal{S}} \subseteq_{rep}^{\ell} \mathcal{S}$  of size at most  $\binom{p+\ell}{p}$  in  $\mathcal{O}\left(\binom{p+\ell}{p} t p^{\omega} + t \binom{p+\ell}{p}^{\omega-1} + \|A_M\|^{\mathcal{O}(1)}\right)$  operations over  $\mathbb{F}$ , where  $\|A_M\|$  denotes the length of  $A_M$  in the input.*

For the purpose of this article, it is enough to know that partition matroids are “representable” [10, Proposition 3.5] and a “truncation” of partition matroids are computable in deterministic polynomial time [9, Theorem 1.3]. This results in Proposition 8, which we will use for our algorithm without giving further definitions of representation and truncation [4, 9, 10].

Let  $\mathcal{F} = \{F_1, F_2, \dots, F_m\}$  be the subsets of  $\mathcal{U}$ ,  $k$  be a positive integer. Since, the sets in  $\mathcal{F}$  are pairwise disjoint, the number of sets in  $\mathcal{F}$  is upper bounded by  $k$ . We call a set  $S \subseteq \mathcal{U}$ , a *potential solution*, if for all  $j \in [\ell]$ ,  $|S \cap B_j| \leq f(B_j)$ . Let

$$\mathcal{S}_i := \{S : S \text{ is a potential solution, } |S| = i \text{ and for all } j \in [i] |S \cap F_j| = 1\}.$$

Given  $\mathcal{S}_i$ , we define  $\mathcal{S}_i^{\text{emb}}$  as  $\{\text{embed}(S) \mid S \in \mathcal{S}_i\}$ . Observe that  $\mathcal{S}_i^{\text{emb}} \subseteq \mathcal{I}$  and each set has size at most  $qi$ . Notice that, each set in  $\mathcal{S}$  has size *exactly*  $i$ , but the same can not be said about the sets in  $\mathcal{S}_i^{\text{emb}}$ . However, since each element occurs in at most  $q$  sets of  $\mathcal{B}$ , we have that each set in  $\mathcal{S}_i^{\text{emb}}$  has size at most  $qi$ .

Our algorithm checks whether  $\mathcal{S}_k$  is non-empty or not. Towards that first observe that  $\mathcal{S}_k$  is non-empty if and only if  $\mathcal{S}_k^{\text{emb}}$  is non-empty. So the testing of non-emptiness of  $\mathcal{S}_k$  boils down to checking whether  $\mathcal{S}_k^{\text{emb}}$  is non-empty or not. We test whether  $\mathcal{S}_k^{\text{emb}}$  is non-empty by computing  $\widehat{\mathcal{S}}_k^{\text{emb}} \subseteq_{\text{rep}}^0 \mathcal{S}_k^{\text{emb}}$  and checking whether  $\widehat{\mathcal{S}}_k^{\text{emb}}$  is non-empty. To argue the correctness of the algorithm, first we have the following observation.

► **Observation 9.**  $\mathcal{S}_k^{\text{emb}} \neq \emptyset$  iff  $\widehat{\mathcal{S}}_k^{\text{emb}} \neq \emptyset$ .

**Proof.** Since  $\widehat{\mathcal{S}}_k^{\text{emb}} \subseteq \mathcal{S}_k^{\text{emb}}$ , the reverse direction is immediate. Now we argue the forward direction. Suppose  $\mathcal{S}_k^{\text{emb}}$ , then it contains some set  $A$ . Note that  $A$  trivially satisfies  $A \cap \emptyset = \emptyset$ . Therefore, since  $\widehat{\mathcal{S}}_k^{\text{emb}} \subseteq_{\text{rep}}^0 \mathcal{S}_k^{\text{emb}}$ , there must exist a set  $\hat{A} \in \widehat{\mathcal{S}}_k^{\text{emb}}$  such that  $\hat{A} \cap \emptyset = \emptyset$ , i.e.,  $\widehat{\mathcal{S}}_k^{\text{emb}} \neq \emptyset$ . ◀

Thus, having computed the representative family  $\widehat{\mathcal{S}}_k^{\text{emb}}$  all we need to do is to check whether it is non-empty. All that remains is an algorithm that computes the representative family  $\widehat{\mathcal{S}}_k^{\text{emb}}$ .

Let  $\mathcal{Z}$  be a family of sets and  $\ell$  be an integer, then  $\mathcal{Z}[\ell]$  is a subset of  $\mathcal{Z}$  that contains *all the sets of*  $\mathcal{Z}$  of size *exactly*  $\ell$ . We describe a dynamic programming based algorithm. Let  $\mathcal{D}$  be an array indexed from integers in  $\{0, 1, \dots, k\}$ . The entry  $\mathcal{D}[i]$  stores the following for all  $j \in \{i, \dots, qi\}$ ,  $\widehat{\mathcal{S}}_i^{\text{emb}}[j] \subseteq_{\text{rep}}^{qk-j} \mathcal{S}_i^{\text{emb}}[j]$ .

We fill the entries in the matrix  $\mathcal{D}$  in the increasing order of indexes. For  $i = 0$ ,  $\mathcal{D}[0] = \emptyset$ . Suppose, we have filled all the entries until the index  $i$ . Then consider the set

$$\mathcal{N}^{i+1} = \{X' = X \cup \text{embed}(\{u\}) : X \in \mathcal{D}[i], u \in F_{i+1}, \text{projection}(X') \text{ is a potential solution}\}$$

We partition sets in  $\mathcal{N}^{i+1}$  based on sizes. Let  $\mathcal{N}^{i+1}[j]$  denote all the sets in  $\mathcal{N}^{i+1}$  of size  $j$ .

▷ **Claim 10.** For all  $j \in \{i+1, \dots, q(i+1)\}$ ,  $\mathcal{N}^{i+1}[j] \subseteq_{\text{rep}}^{qk-j} \mathcal{S}_{i+1}^{\text{emb}}[j]$ .

**Proof.** Let  $S \in \mathcal{S}_{i+1}^{\text{emb}}[j]$  and  $Y$  be a set of size at most  $qk - j$  (which is essentially an independent set of the matroid  $\mathcal{M} = (\widetilde{U}, \mathcal{I})$ ) such that  $S \cap Y = \emptyset$  and  $S \cup Y \in \mathcal{I}$ . We will show that there exists a set  $S' \in \mathcal{N}^{i+1}[j]$  such that  $S' \cap Y = \emptyset$  and  $S' \cup Y \in \mathcal{I}$ . This will imply the desired result. Since  $S \in \mathcal{S}_{i+1}^{\text{emb}}[j]$  there exists an element  $u \in F_{i+1}$  such that

$$S = (S \setminus \text{embed}(\{u\})) \cup \text{embed}(\{u\}).$$

Let  $S_i = (S \setminus \text{embed}(\{u\}))$ . Since,  $S$  is an independent set of the matroid  $\mathcal{M} = (\widetilde{U}, \mathcal{I})$ , we have that  $S_i$  is an independent set of the matroid  $\mathcal{M} = (\widetilde{U}, \mathcal{I})$  (hereditary property). Further,  $|\text{projection}(S_i)| = i$  and  $\text{projection}(S_i)$  is a hitting set for  $F_1, \dots, F_i$ . This implies that  $S_i \in \mathcal{S}_i^{\text{emb}}$ . Let  $Y_i = Y \cup \text{embed}(\{u\})$ . Notice that since  $S \cap Y = \emptyset$  and  $S \cup Y \in \mathcal{I}$ , we have that  $Y_i$  is an independent set and  $S_i \cup Y_i = S \cup Y \in \mathcal{I}$ . Let  $|S_i| = j'$ . Then, we know that  $\mathcal{D}[i]$  contains  $\widehat{\mathcal{S}}_i^{\text{emb}}[j'] \subseteq_{\text{rep}}^{qk-j'} \mathcal{S}_i^{\text{emb}}[j']$ . This implies that there exists a set  $S'_i \in \widehat{\mathcal{S}}_i^{\text{emb}}[j']$  such that  $S'_i \cup Y_i \in \mathcal{I}$ . This implies that  $S'_i \cup \text{embed}(\{u\})$  is in  $\mathcal{N}^{i+1}$ . Further, since  $|S| = \sum_{x \in \text{projection}(S)} |\text{embed}(\{x\})|$ , we have that  $|S'_i \cup \text{embed}(\{u\})| = |S_i| = j$ . This implies that  $S'_i \cup \text{embed}(\{u\})$  is in  $\mathcal{N}^{i+1}[j]$ . Thus, we can take  $S' = S'_i \cup \text{embed}(\{u\})$ . This completes the proof. ◀

We fill the entry for  $\mathcal{D}[i+1]$  as follows. We first compute  $\mathcal{N}^{i+1}$ . Observe that the sets in  $\mathcal{N}^{i+1}$  have sizes ranging from  $i+1$  to  $q(i+1)$ . Now we apply Proposition 8 on each of  $\mathcal{N}^{i+1}[j]$ ,  $j \in \{i+1, \dots, q(i+1)\}$ , and compute  $qk-j$  representative. That is, we compute  $\widehat{\mathcal{N}}^{i+1}[j] \subseteq_{rep}^{qk-j} \mathcal{N}^{i+1}[j]$ . We set

$$\mathcal{D}[i+1] = \bigcup_{j=i+1}^{q(i+1)} \widehat{\mathcal{N}}^{i+1}[j].$$

Observe that the number of sets in  $\mathcal{D}[i]$  of size  $j$  is upper bounded by  $\binom{q(k-i)+j}{j} \leq \binom{qk}{di} \leq 2^{\mathcal{O}(qk)}$ . Hence, the time taken to compute  $\mathcal{D}[i]$  is upper bounded by  $2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$ . Thus, the time taken to compute  $\mathcal{D}[i+1]$  requires at most  $qk$  invocations of Proposition 8. This itself takes  $2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$  time. This completes the proof, resulting in the following result.

► **Theorem 2.** *Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. Then, FAIR HITTING SET can be solved in time  $2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$  time, when every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and any pair of sets in  $\mathcal{F}$  are pairwise disjoint. Further, FAIR HITTING SET admits a kernel of size  $\mathcal{O}(kq^2 \binom{kq}{q} \log k)$ .*

Theorem 2 can be generalized to the scenario where every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and at most  $d$  sets in  $\mathcal{F}$ . Observe that if each element appear in at most  $d$  sets of  $\mathcal{F}$ , then the total number of sets that a subset of size  $k$  of  $\mathcal{U}$  can hit is upper bounded by  $dk$ , else we immediately return that given instance is a NO-instance. Let  $S = \{u_1, \dots, u_k\}$  be a hypothetical solution to our problem. Now, with the help of  $S$ , we partition  $\mathcal{F}$  as follows. Let  $\mathcal{F}_i$  denote all sets in  $\mathcal{F}$  that contain  $u_i$  and none of  $\{u_1, \dots, u_{i-1}\}$ . Clearly,  $\mathcal{F}_i$ ,  $i \in [k]$ , partitions  $\mathcal{F}$ . Now we can design a dynamic programming algorithm similar to the one employed in Theorem 2, where in each iteration we grow our representative family by elements that only hit sets in  $\mathcal{F}_i$  and not in  $\mathcal{F}_j$ ,  $j > i$ . This will result in the following theorem, whose proof can be found in the full version.

► **Theorem 3.** *Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. Then, FAIR HITTING SET can be solved in time  $k^{\mathcal{O}(dk)} 2^{\mathcal{O}(qk)} n^{\mathcal{O}(1)}$  time, when every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and at most  $d$  sets in  $\mathcal{F}$ .*

### 3.3 A Kernel for a Special Case of Fair Hitting Set using Matroids

In this section we design a polynomial kernel for the same special case of FAIR HITTING SET, that we considered in the last section. Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET and assume that every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and any pair of sets in  $\mathcal{F}$  are pairwise disjoint. To design our kernel we will again use Lemma 6 that says that an input  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  is a yes-instance if and only if there exists an independent set  $X \in \mathcal{I}$  of the matroid  $\mathcal{M} = (\tilde{\mathcal{U}}, \mathcal{I})$  such that  $|\text{projection}(X)| \leq k$  and  $\text{projection}(X)$  is a hitting set for  $\mathcal{F}$ .

Let  $\mathcal{F} = \{F_1, F_2, \dots, F_m\}$  be the subsets of  $\mathcal{U}$ , and  $k$  be a positive integer. Since, the sets in  $\mathcal{F}$  are pairwise disjoint, the number of sets in  $\mathcal{F}$  is upper bounded by  $k$ . In particular, we assume that  $m = k$ . We define  $\mathcal{F}_i^{\text{emb}} = \{\text{embed}(\{u\}) \mid u \in F_i\}$ . For our kernel we apply the following reduction rules. We start with some simple reduction rules.

- **Reduction Rule 1.** Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET.
- If there exists an element  $u \in \mathcal{U}$ , such that  $u$  does not appear in any sets in  $\mathcal{F}$  then delete it from  $\mathcal{U}$  and all the sets in  $\mathcal{B}$  that it appears in.
  - If there exists a set  $B \in \mathcal{B}$  such that  $B = \emptyset$ , then delete  $B$ , and take  $f$  as the restriction of  $f$  on  $\mathcal{B} \setminus \{B\}$ .

- If there exists a set  $B \in \mathcal{B}$  such that  $f(B) = 0$ , then we do as follows:  $\mathcal{U} := \mathcal{U} \setminus \{B\}$ ; delete all the elements of  $B$  from all the sets in  $\mathcal{B}$  and  $\mathcal{F}$  that it appears in. If some set in  $\mathcal{F}$  becomes empty then return a trivial NO-instance. Else, take  $f$  as the restriction of  $f$  on  $\mathcal{B} \setminus \{B\}$  and keep the integer  $k$  unchanged.

Soundness of Reduction Rule 1 is obvious and hence omitted. The next reduction rule is the main engine of our kernel.

► **Reduction Rule 2.** Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET. If there exists a pair of integers  $i \in [k]$  and  $j \in [q]$  such that  $|\mathcal{F}_i^{\text{emb}}[j]| > \binom{kq}{j}$ , then do as follows. Compute  $\widehat{\mathcal{F}}_i^{\text{emb}}[j] \subseteq_{\text{rep}}^{qk-j} \mathcal{F}_i^{\text{emb}}[j]$ . Let  $F \in \mathcal{F}_i^{\text{emb}}[j]$  that do not appear in  $\widehat{\mathcal{F}}_i^{\text{emb}}[j]$ . Then obtain a reduced instance as follows.

- $\mathcal{U} := \mathcal{U} \setminus \text{projection}(F)$
- Delete  $\text{projection}(F)$  from all the sets in  $\mathcal{B}$  and  $\mathcal{F}$  that it appears in.
- The function  $f$  and  $k$  remains the same.

► **Lemma 11.** *Reduction Rule 2 is sound.*

**Proof.** Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET, and let  $(\mathcal{U}', \mathcal{F}', \mathcal{B}', f : \mathcal{B}' \rightarrow \mathbb{N}, k)$  be the reduced instance, after an application of Reduction Rule 2. It is easy to see that a solution to the reduced instance can directly be lifted to the input instance. Thus, we focus on forward direction.

In the forward direction, let  $S$  be a solution to  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$ . Then, by Lemma 6, it implies that  $\text{embed}(S) \in \mathcal{I}$  (of the matroid  $\mathcal{M} = (\widetilde{\mathcal{U}}, \mathcal{I})$ ) and  $|S| \leq k$  and  $S = \text{projection}(\text{embed}(S))$  is a hitting set for  $\mathcal{F}$ . Let  $u = \text{projection}(F)$ . Then, we have that  $\mathcal{U} := \mathcal{U} \setminus \{u\}$ . If  $u \notin S$ , then  $S$  is also the solution to  $(\mathcal{U}', \mathcal{F}', \mathcal{B}', f : \mathcal{B}' \rightarrow \mathbb{N}, k)$ . So we assume that  $u \in S$ .

Observe that  $F = \text{embed}(\{u\})$ ,  $|F| = j$ , and  $u$  belongs to  $F_i$ . Further, since every set in  $\mathcal{F}$  are pairwise disjoint we have that the only job of  $u$  is to hit the set  $F_i$ . Consider,  $Y = \text{embed}(S) \setminus \text{embed}(\{u\})$ . Since,  $\text{embed}(S) \in \mathcal{I}$ , we have that  $Y \in \mathcal{I}$  (hereditary property of the matroid), and the size of  $Y$  is upper bounded by  $qk - j$ . The last assertion follows from the fact that for any element  $v \in \mathcal{U}$ , the size of  $\text{embed}(\{v\})$  is upper bounded by  $q$  and  $|\text{embed}(S)| = \sum_{x \in S} |\text{embed}(\{x\})| \leq qk$ . This implies that there exists  $F' \in \widehat{\mathcal{F}}_i^{\text{emb}}[j] \subseteq_{\text{rep}}^{qk-j} \mathcal{F}_i^{\text{emb}}[j]$  such that  $Y \cup \{F'\} \in \mathcal{I}$ . Since,  $|\text{projection}(\text{embed}(S))| \leq k$ , we have that  $|\text{projection}(Y)| \leq k - 1$ . Thus,  $|\text{projection}(Y \cup \{F'\})| \leq k$ . Now we need to show that  $\text{projection}(Y \cup \{F'\})$  is a hitting set for  $\mathcal{F}$ . This follows from the fact that  $u' = \text{projection}(F') \in F_i$ . In other words, we have shown that  $S' = S \setminus \{u\} \cup \{u'\}$  is a desired hitting set for  $\mathcal{F}$ . This concludes the proof. ◀

Finally, we get the following kernel.

► **Theorem 12.** *Let  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  be an instance of FAIR HITTING SET such that every element in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$  and any pair of sets in  $\mathcal{F}$  are pairwise disjoint. Then, FAIR HITTING SET admits a kernel of size  $\mathcal{O}(kq^2 \binom{kq}{q} \log k)$ .*

**Proof.** For our algorithm we apply Reduction Rules 1 and 2 exhaustively. If any application of these rules return that the input is a NO-instance, we return the same. The correctness of the algorithm follows from the correctness of Reduction Rules 1 and 2. Further it is clear that the algorithm runs in polynomial time. What remains to show is that the reduced instance is upper bounded by the claimed function.

For convenience we assume that the reduced instance is also denoted by  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$ . Since, Reduction Rule 2 is not applicable we have that each set  $F_i$ ,  $i \in [k]$ , is upper bounded by  $\sum_{j=1}^q \binom{kq}{j} \leq q \binom{kq}{q}$ . This implies that  $|\mathcal{U}| \leq kq \binom{kq}{q}$ . Further, since every element

in  $\mathcal{U}$  appears in at most  $q$  sets in  $\mathcal{B}$ , we have that the number of non-empty sets in  $\mathcal{B}$  is upper bounded by  $q|\mathcal{U}| \leq kq^2 \binom{kq}{q}$ . Since, Reduction Rule 1 is not applicable we have that there are no empty-sets and hence  $|\mathcal{B}| \leq kq^2 \binom{kq}{q}$ . Further, to represent the function  $f$  we need at most  $\mathcal{O}(|\mathcal{B}| \log k) \leq \mathcal{O}(kq^2 \binom{kq}{q} \log k)$  bits. This completes the proof.  $\blacktriangleleft$

#### 4 Reduction from $K_{d,d}$ -free $G_{\mathcal{U},\mathcal{F}}$ to Bounded Frequency in $\mathcal{F}$

In Section 4.1, we consider a special case of the above setting: (1)  $G_{\mathcal{U},\mathcal{F}}$  is  $K_{d,d}$ -free, and (2) each element in  $\mathcal{U}$  has frequency at most  $q$  in  $\mathcal{B}$ . For this case, we design a polynomial kernel. For the sake of brevity, we give a detailed, yet informal, overview of the kernelization, and defer the formal details to the full version.

A part of the kernelization procedure can also be used to bound the frequency of an element in  $\mathcal{F}$  by a function of  $k$  and  $d$ . We give an alternate, self-contained proof of this theorem in the full version. This reduction is used as the first step in some of our results, such as Section 5.

##### 4.1 Polynomial Kernel for $K_{d,d}$ -free $G_{\mathcal{U},\mathcal{F}}$ and Bounded Frequency in $\mathcal{B}$

Consider an input  $(\mathcal{U}, \mathcal{F}, \mathcal{B}, f : \mathcal{B} \rightarrow \mathbb{N}, k)$  of FAIR HITTING SET. In this section we design a polynomial kernel for FAIR HITTING SET problem when  $G_{\mathcal{U},\mathcal{F}}$  is  $K_{d,d}$ -free and frequency of each element in  $\mathcal{B}$  is at most  $q$ . We fix  $d$  and  $q$  for the rest of the section. Without loss of generality we assume that  $d \geq 2$ ,  $k \geq 2$ . We also assume that we do not have multisets in  $\mathcal{F}$  and  $\mathcal{B}$ .

Under these assumptions, the kernelization algorithm consists of two phases. In the first phase we apply some reduction rules to bound the size of  $|\mathcal{F}|$ . In the second phase, we use the partition matroid  $\mathcal{M} = (\tilde{\mathcal{U}}, \mathcal{I})$ , as defined in Section 3.1 using  $\mathcal{B}$  to design a reduction rule to bound the number of elements. Now we discuss each of these phases in more detail.

###### Phase 1

We first define the following easy reduction rules that handle some of the easy cases

- We can delete an empty set from  $\mathcal{B}$  without affecting the instance.
- We can delete an element  $u \in \mathcal{U}$  that is not contained in any set in  $\mathcal{F}$ .
- If there exists a set  $B \in \mathcal{B}$  with  $f(B) = 0$ , then we consider two cases.
  1. If there exists some  $F \in \mathcal{F}$  such that  $F \subseteq B$ , then we have a no-instance.
  2. Otherwise, we can delete  $B$  from  $\mathcal{B}$ , and delete all elements of  $B$  from the universe  $\mathcal{U}$  as well as the corresponding sets in  $\mathcal{F}$ .

Each such rule can be implemented in polynomial time. We emphasize that these reduction rules are repeatedly applied in this order, after each application of subsequent rules.

Next, we consider the following case. If there exists an element  $u \in \mathcal{U}$  contained in at least  $dk^{d-1}$  sets of  $\mathcal{F}$ , then in *polynomial time* we can find a non-empty set  $X \subseteq \mathcal{U}$  of size at most  $d-1$  with the following properties: (1)  $X$  intersects with *every* fair hitting set of size at most  $k$ , and (2) the number of sets in  $\mathcal{F}$  that contain  $X$  as a subset is *large*, i.e., at least  $dk^{d-1}$ . This is where we crucially use the fact that  $G_{\mathcal{U},\mathcal{F}}$  is  $K_{d,d}$ -free. Now, we can use such a set  $X$  to reduce the size of instance, by either finding some  $u \in \mathcal{U}$ , or some  $F_i \in \mathcal{F}$  that can be deleted without affecting the instance.

Note that since we reduce the size of instance in each application of the rule, this rule is applicable only polynomially many times. Furthermore, when the rule is not applicable, it follows that every element of  $\mathcal{U}$  is contained in at most  $dk^{d-1}$  sets of  $\mathcal{F}$ . Here, we observe that

we did not make any specific assumptions about the hypergraph  $(\mathcal{U}, \mathcal{B})$ , except that we may delete an element of  $\mathcal{U}$  or a set from  $\mathcal{B}$ . Thus, the resulting hypergraph is a sub-hypergraph of the original hypergraph  $(\mathcal{U}, \mathcal{B})$ . Thus, if  $G_{\mathcal{U}, \mathcal{B}}$  satisfies some hereditary property  $\Pi$ , then the resulting incidence graph continues to satisfy  $\Pi$ . Thus, phase 1 constitutes a proof of .

At this step, if the number of sets in  $\mathcal{F}$  is larger than  $dk^d$ , no subset of size  $k$  can hit all sets in  $\mathcal{F}$ . Thus, we simply conclude that such an instance is a no-instance of the problem. This concludes Phase 1.

Suppose in the original instance  $G_{\mathcal{U}, \mathcal{B}}$  satisfied that the frequency of every element in  $\mathcal{B}$  is at most  $q$ . Then, at this step, we can use an approach similar to Theorem 3 to design an FPT algorithm that runs in time  $d^k k^{kd} \cdot 2^{\mathcal{O}(kq)} \cdot |\mathcal{I}|^{\mathcal{O}(1)}$ . Now, we proceed to Phase 2, where we further reduce the size of an instance to design a kernel under the assumption when the frequency in  $\mathcal{B}$  is bounded by  $q$ .

## Phase 2

We observe that the number of elements in  $\mathcal{U}$  with frequency at least  $d$  in  $\mathcal{F}$  is bounded by  $\binom{|\mathcal{F}|}{d} \cdot d$ . Let  $\mathcal{U}'$  denote the elements that are contained in at most  $d - 1$  sets of  $\mathcal{F}$ . We define equivalence classes of  $\mathcal{U}'$ , such that all elements in the same class belong to all the sets of  $\mathcal{Y}$ , where  $\mathcal{Y} \subseteq \mathcal{F}$  is a sub-family of size at most  $d - 1$ . Let us denote such a subset by  $\text{ExactNbr}(\mathcal{Y})$ .

We use the matroid-based techniques developed in the previous section, in order to reduce the number of distinct elements in  $\text{ExactNbr}(\mathcal{Y})$  that we need to remember. In particular, we show that for every  $\mathcal{Y} \subseteq \mathcal{F}$ , we only need to remember at most  $\binom{kq}{q}$  distinct elements, and we may delete the rest in a careful manner.

Thus, at the end, we have the following. The number of elements with degree (i.e., frequency) at most  $d - 1$  in  $G_{\mathcal{U}, \mathcal{F}}$  is bounded by  $d \cdot \binom{|\mathcal{F}|}{d} \cdot \binom{kq}{q}$ . Accounting for the elements with large degree, the total number of elements in  $\mathcal{U}$  is bounded by  $\mathcal{O}(k^{\mathcal{O}(d^2+q)} d^d q^q)$ . Since each element has degree at most  $q$  in  $G_{\mathcal{U}, \mathcal{B}}$ , we can also bound the number of sets in  $\mathcal{B}$ . Observe that each of our reduction rules can be applied in polynomial time and only polynomially many times. Thus, we prove the following theorem.

► **Theorem 4.** *Given an instance  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f, k)$  of FAIR HITTING SET, such that  $G_{\mathcal{U}, \mathcal{F}}$  is  $K_{d,d}$ -free, and the frequency of each element in  $\mathcal{B}$  is bounded by  $q$ , one can find an equivalent instance  $\mathcal{I}' = (\mathcal{U}', \mathcal{F}', \mathcal{B}', f', k')$  of FAIR HITTING SET in polynomial time, such that  $|\mathcal{U}'| = \mathcal{O}(k^{d^2+q} d^d q^q)$ ,  $|\mathcal{F}'| \leq dk^d$ , and  $|\mathcal{B}'| = \mathcal{O}(k^{d^2+q} \cdot d^d q^{q+1})$ , where  $d$  and  $q$  are assumed to be constants.*

## 5 Parameterization by $k + d$ when $G_{\mathcal{U}, \mathcal{B}}$ is nowhere dense and $G_{\mathcal{U}, \mathcal{F}}$ is $K_{d,d}$ -free

We give a brief overview of the following theorem, a formal proof can be found in the full version.

► **Theorem 5.** *Let  $\mathcal{G}$  be a nowhere dense graph class. Let  $\mathcal{I} = (\mathcal{U}, \mathcal{F}, \mathcal{B}, f, k)$  be an instance of FAIR HITTING SET such that the incidence graph  $G := G_{\mathcal{U}, \mathcal{B}} \in \mathcal{G}$ , and  $G_{\mathcal{U}, \mathcal{F}}$  is  $K_{d,d}$ -free for some  $d \geq 1$ . Then, one can solve FAIR HITTING SET on  $\mathcal{I}$  in time  $h(k, d) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$ , for some function  $h(\cdot, \cdot)$ .*

First, we reduce the given instance, where  $G_{\mathcal{U}, \mathcal{F}}$  is  $K_{d,d}$ -free, to an instance where the size of  $|\mathcal{F}|$  is bounded by  $d \cdot k^d$ . Thus, it suffices to design an FPT algorithm parameterized by  $k, d$  and  $m := |\mathcal{F}|$ . The rest of the section focuses on designing such an algorithm. We

prove Theorem 5 by reducing the problem to FO model checking on  $G$ .<sup>1</sup> Due to lack of space, the necessary definitions and background pertaining to nowhere dense graph classes and first-order logic is given in the full version.

We reduce the problem of deciding whether  $\mathcal{I}$  is a yes-instance of FAIR HITTING SET to the problem of First-Order (FO) model checking. We first “guess” the structure of a hypothetical solution (i.e.,  $S \subseteq \mathcal{U}$  such that  $S$  is a fair hitting set for  $\mathcal{F}$ ), if any. More specifically, we guess the exact size  $k'$  of a hypothetical solution, and the exact subset of  $\mathcal{F}$  that is hit by each element of the solution. Note that there are at most  $2^{\mathcal{O}(km)}$  possible guesses. For each such guess, we create a first-order logic formula that is true if and only if such a solution is a fair hitting set, i.e., it hits all the sets in  $F$ , and for each  $B_j \in \mathcal{B}$ , the size of the intersection of  $B_j$  with the solution is at most  $f(B_j)$ . We show that the size of the formula is upper bounded by a function of  $k$  and  $|\mathcal{F}|$ . Finally, since model checking of first-order logic formulas can be decided in polynomial time on nowhere dense graphs, the theorem follows.

---

### References

- 1 Johannes Blum, Yann Disser, Andreas Emil Feldmann, Siddharth Gupta, and Anna Zych-Pawlewicz. On sparse hitting sets: from fair vertex cover to highway dimension. *CoRR*, abs/2208.14132, 2022. doi:10.48550/arXiv.2208.14132.
- 2 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 3 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 4 Fedor V. Fomin, Daniel Lokshantov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.
- 5 Pallavi Jain, Lawqueen Kanesh, and Pranabendu Misra. Conflict free version of covering problems on graphs: Classical and parameterized. *Theory Comput. Syst.*, 64(6):1067–1093, 2020. doi:10.1007/s00224-019-09964-6.
- 6 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- 7 Dusan Knop, Tomáš Masarík, and Tomáš Toufar. Parameterized complexity of fair vertex evaluation problems. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 33:1–33:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.33.
- 8 Li-Shin Lin and Sartaj Sahni. Fair edge deletion problems. *IEEE Trans. Computers*, 38(5):756–761, 1989. doi:10.1109/12.24280.
- 9 Daniel Lokshantov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018. doi:10.1145/3170444.
- 10 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.
- 11 Tomáš Masarík and Tomáš Toufar. Parameterized complexity of fair deletion problems. *Discret. Appl. Math.*, 278:51–61, 2020. doi:10.1016/j.dam.2019.06.001.

---

<sup>1</sup> More accurately, we reduce to FO model checking on a *colored* version of  $G$ , where the vertices of  $G$  are partitioned into a number of *color classes*, and in the FO formula, we may require that a vertex belongs to a specific color class. Formal definitions can be found in the full version.

# Parameterized Approximation Scheme for Feedback Vertex Set

Satyabrata Jana ✉ 

Institute of Mathematical Sciences, HBNI, Chennai, India

Daniel Lokshtanov ✉

Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA, USA

Soumen Mandal ✉ 

Department of Mathematics, IIT Delhi, New Delhi, India

Ashutosh Rai ✉

Department of Mathematics, IIT Delhi, New Delhi, India

Saket Saurabh ✉

Institute of Mathematical Sciences, HBNI, Chennai, India

University of Bergen, Bergen, Norway

---

## Abstract

---

FEEDBACK VERTEX SET (FVS) is one of the most studied vertex deletion problems in the field of graph algorithms. In the decision version of the problem, given a graph  $G$  and an integer  $k$ , the question is whether there exists a set  $S$  of at most  $k$  vertices in  $G$  such that  $G - S$  is acyclic. It is one of the first few problems which were shown to be NP-complete, and has been extensively studied from the viewpoint of approximation and parameterized algorithms. The best-known polynomial time approximation algorithm for FVS is a 2-factor approximation, while the best known deterministic and randomized FPT algorithms run in time  $\mathcal{O}^*(3.460^k)$  and  $\mathcal{O}^*(2.7^k)$  respectively.<sup>1</sup>

In this paper, we contribute to the newly established area of parameterized approximation, by studying FVS in this paradigm. In particular, we combine the approaches of parameterized and approximation algorithms for the study of FVS, and achieve an approximation guarantee with a factor better than 2 in randomized FPT running time, that improves over the best known parameterized algorithm for FVS. We give three simple randomized  $(1 + \epsilon)$  approximation algorithms for FVS, running in times  $\mathcal{O}^*(2^{\epsilon k} \cdot 2.7^{(1-\epsilon)k})$ ,  $\mathcal{O}^*\left(\left(\left(\frac{4}{1+\epsilon}\right)^{(1+\epsilon)} \cdot \left(\frac{\epsilon}{3}\right)^\epsilon\right)^k\right)$ , and  $\mathcal{O}^*(4^{(1-\epsilon)k})$  respectively for every  $\epsilon \in (0, 1)$ . Combining these three algorithms, we obtain a factor  $(1 + \epsilon)$  approximation algorithm for FVS, which has better running time than the best-known (randomized) FPT algorithm for every  $\epsilon \in (0, 1)$ . This is the first attempt to look at a parameterized approximation of FVS to the best of our knowledge. Our algorithms are very simple, and they rely on some well-known reduction rules used for arriving at FPT algorithms for FVS.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Feedback Vertex Set, Parameterized Approximation

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.56

**Funding** *Soumen Mandal*: Supported by Council of Scientific and Industrial Research, India.

*Ashutosh Rai*: Supported by Science and Engineering Research Board (SERB) Grant SRG/2021/002412.

*Saket Saurabh*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).

---

<sup>1</sup> here the  $\mathcal{O}^*$ () notations hides the polynomial factors in the running time of the algorithm.



## 1 Introduction

Vertex deletion problems are one of the most basic and well-studied classes of graph editing problems. In the decision version of these problems, given a graph  $G$  and integer  $k$ , we are asked whether we can delete at most  $k$  vertices from  $G$  such that the resulting graph satisfies certain properties. When we want the resulting graph to be empty or acyclic, then the vertex deletion problem corresponds to the well-known VERTEX COVER and FEEDBACK VERTEX SET problems respectively. These were two of the first few problems to be shown NP-complete and appear in Karp’s list of 21 NP-complete problems. In this paper, we concentrate on the FEEDBACK VERTEX SET problem, in the realm of parameterized approximation. The problem is formally defined as follows.

FEEDBACK VERTEX SET (FVS)	<b>Parameter:</b> $k$
<b>Input:</b> A graph $G = (V, E)$ and a positive integer $k$ .	
<b>Question:</b> Does there exist a set $S \subseteq V(G)$ such that $ S  \leq k$ and $G - S$ is acyclic?	

Parameterized approximation combines the fields of parameterized and approximation algorithms, where we try to get the best of both worlds by achieving a better guarantee than that of the best-known approximation algorithm (running in polynomial time), while the algorithm takes FPT time, but beats the best known exact algorithm for the problem. Such algorithms are termed “parameterized approximation” algorithms in the literature. An even more ambitious goal can be to get a parameterized approximation scheme, where for every  $\epsilon \in (0, \alpha - 1)$  (where  $\alpha$  is the best-known approximation factor for polynomial-time algorithms), we get a factor  $(1 + \epsilon)$  approximation algorithm, that runs in FPT time and is faster than the best known parameterized algorithm for the problem.

Probably the best example for this approach is the MIN  $k$ -CUT problem (delete a minimum number of edges to get at least  $k$  connected components), which is not expected to be FPT when parameterized by  $k$  since it is W[1]-hard. The best-known approximation factor, that is possible in polynomial time, for this is 2 [20]. Gupta et al. [11] showed that we can get an approximation ratio better than 2 by allowing FPT running time, and in a recent breakthrough result, Lokshtanov et al. [19] designed a *parameterized approximation scheme* for the problem, which means that for every  $\epsilon > 0$ , there is an algorithm running in FPT time which gives a factor  $(1 + \epsilon)$  approximation for MIN  $k$ -CUT. There are a few of lower bounds results in the field as well [3, 5, 6, 17], which show that this approach does not work for certain problems.

In addition to problems that are not expected to be FPT, researchers have also studied problems that are FPT from the lens of parameterized approximation. For these problems, we try to get a better approximation factor (than what is known in polynomial time) by allowing faster FPT running time (as compared to the best FPT algorithm for the problem). This approach has been applied to many problems including VERTEX COVER,  $d$ -HITTING SET [10, 4, 15], classical cut problems like DIRECTED-FVS, MULTICUT [18] etc. For a comprehensive overview of the current state of parameterized approximation, we refer to the recent survey by Feldmann et al. [9], as well as the surveys conducted by Kortsarz [14] and Marx [21].

In this paper, we look at FEEDBACK VERTEX SET (FVS), which is one of the most studied problems in the field of parameterized complexity, from the parameterized approximation lens. For FVS, the best-known approximation factor that can be achieved in polynomial time is 2 [1], and this is the best approximation factor that we can hope for FVS in polynomial time under the famous Unique Games Conjecture [13]. On the other side, the fastest known deterministic and randomized FPT algorithms for FVS run in times  $\mathcal{O}^*(3.460^k)$  [12] and

$\mathcal{O}^*(2.7^k)$  [16] respectively. We want to design a parameterized approximation scheme for FVS. This turns out to be a harder task than designing such schemes for problems like VERTEX COVER. For VERTEX COVER, there is a simple branching algorithm, that picks an edge and branches on its endpoints, but no such simple algorithm exists for FVS because the forbidden subgraph for FVS, a cycle, can be very large. For designing a good parameterized approximation scheme for VERTEX COVER or  $d$ -HITTING SET, there are two ways to achieve it: i) pick disjoint copies of edges or sets in the solution and then run the fastest FPT algorithm on the remaining instance, and ii) do branching for some steps, and then run an approximation algorithm on the remaining instance. These seem difficult to achieve for FVS because the cycles can be very big, and hence the approximation guarantee in i) and the branching factor in ii) are not bounded. In this paper, we overcome these difficulties using ideas from known randomized algorithms and obtain the following result.

► **Theorem 1.** *There exists a randomized algorithm that, given an instance  $(G, k)$  of FVS and  $\epsilon \in (0, 1)$ , either reports a failure or finds a feedback vertex set in  $G$  of size at most  $(1 + \epsilon)k$  in time*

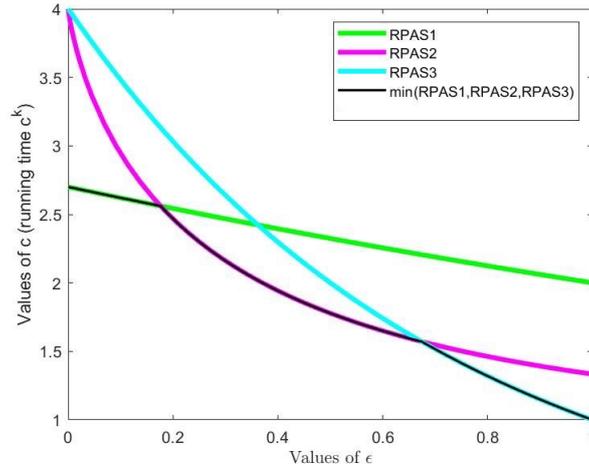
$$\mathcal{O}^* \left( \min \left\{ 2^{\epsilon k} \cdot 2.7^{(1-\epsilon)k}, \left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^k, 4^{(1-\epsilon)k} \right\} \right).$$

Moreover, if  $G$  has a feedback vertex set of size at most  $k$ , the algorithm returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $1/e$ .

**Our methods.** For proving Theorem 1, we make use of two randomized algorithms, the first being one of the oldest algorithms for FVS that runs in time  $\mathcal{O}^*(4^k)$  [2], and the other being the currently best known randomized algorithm for FVS running in time  $\mathcal{O}^*(2.7^k)$  [16]. We give three different algorithms to get the running time mentioned in Theorem 1, where each of the three algorithms outperforms the other two in some range of  $\epsilon$  in the interval  $[0, 1]$ . For a clearer view, we provide a graph depicting the running times of the three algorithms for different values of  $\epsilon$  in Figure 1. We observe that for every  $\epsilon \in (0, 1)$ , the algorithm of Theorem 1 gives a randomized  $(1 + \epsilon)$  approximation and runs in time better than  $\mathcal{O}^*(2.7^k)$ . All our algorithms make use of some simple reduction rules for FVS, which have been extensively applied to obtain FPT algorithms for the problem. The useful property of the rules is that if none of them are applicable, then the graph has a minimum degree of at least 3.

The first algorithm that we design uses the property that in a graph with a minimum degree of at least 3, at least half the edges are incident on any feedback vertex set of the graph [2]. It picks some edges randomly, adds the endpoints of the edges to the solution, and then it runs the  $\mathcal{O}^*(2.7^k)$  time algorithm of [16] on the remaining graph. This gives an algorithm running in time  $\mathcal{O}^*(2.7^{(1-\epsilon)k})$  and succeeding (giving a factor  $(1 + \epsilon)$  approximation for every  $\epsilon \in (0, 1)$ ) with probability  $c \cdot 2^{-\epsilon k}$  for some  $c \geq \frac{1}{2}$ . We repeat this algorithm  $\frac{1}{c} \cdot 2^{\epsilon k}$  times to get a constant probability of success, and the final running time is  $\mathcal{O}^*(2^{\epsilon k} \cdot 2.7^{(1-\epsilon)k})$ .

The second algorithm that we design also uses the same property, but it picks one of the endpoints randomly from a randomly picked edge in the solution (instead of both endpoints in the first algorithm). We keep doing that till we either exhaust our budget or the graph becomes acyclic. Using the techniques of [15], we show that this algorithm succeeds with probability  $\left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^{-k}$ . Repeating this  $\left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^k$  times, we get constant success probability. The running time of this algorithm is better than the first algorithm for  $\epsilon \in (0.176, 1)$ .



■ **Figure 1** A graph showing running times of the three algorithms, which we name RPAS1, RPAS2, and RPAS3.

The third algorithm, instead of picking the vertices like the second algorithm does till the budget is exhausted, picks vertices up to a certain threshold, and then applies the 2-approximation algorithm of [1] on the remaining graph. This algorithm succeeds with probability  $4^{-(1-\epsilon)k}$ , which gives the running time of  $\mathcal{O}^*(4^{(1-\epsilon)k})$  for constant success probability. This algorithm performs better than the first two algorithms for  $\epsilon \in (0.674, 1)$ .

The main purpose of this article is to put forward a proof of concept for designing FPT approximation algorithms for problems for which the forbidden sets are of unbounded size. All our algorithms are randomized, and getting a deterministic FPT approximation scheme for FVS is an interesting open problem.

## 2 Preliminaries

In this section, we give the notations and definitions, along with some known results and reduction rules which are used in the paper.

For a graph  $G = (V, E)$ , we denote the set of vertices of the graph by  $V(G)$  and the set of edges of the graph by  $E(G)$ . 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  $\{\{u, v\} \in E(G) : u, v \in S\}$  and the subgraph obtained after deleting  $S$  (and the edges incident to the vertices in  $S$ ) is denoted as  $G - S$ . We say a graph  $G$  is acyclic if there is no cycle in the graph. For ease of notation, we will use  $uv$  to denote an edge of a graph instead of  $\{u, v\}$ . We denote the degree of a vertex  $v \in V(G)$  as  $d(v)$  and it is equal to the number of edges incident on  $v$ . In case of a self-loop on  $v$ , the self-loop contributes 2 to  $d(v)$ . The minimum degree of a graph  $G$  is denoted as  $\delta(G)$  and it is defined as  $\delta(G) = \min\{d(v) : v \in V(G)\}$ . For a graph  $G$ , a set  $S \subseteq V(G)$  is called a *feedback vertex set* of  $G$  if  $G - S$  is acyclic. An instance  $(G, k)$  of FVS is said to be a **Yes** instance if there is a feedback vertex set of size at most  $k$  in  $G$ . Given a **Yes** instance  $(G, k)$  of FVS and an  $\epsilon \in (0, 1)$  as an input, we say that an algorithm  $\mathcal{A}$  *succeeds* if it outputs a feedback vertex set of  $G$  of size at most  $(1 + \epsilon)k$ . We denote the size of a minimum-sized feedback vertex set of a graph  $G$  by  $\text{fvs}(G)$ .

Let us state the following reduction rules that we will use in this paper to make the minimum degree of the graph at least 3.

1. [8] If there is a self-loop at a vertex  $v$ , delete  $v$  from the graph and decrease  $k$  by 1.
2. [8] If there is an edge of multiplicity larger than 2, reduce its multiplicity to 2.
3. [8] If there is a vertex  $v$  of degree at most 1, delete  $v$ .
4. [8] If there is a vertex  $v$  of degree 2, delete  $v$  and connect its two neighbors by a new edge.

If none of the above four reduction rules are applicable to a graph  $G$  then we can assume that  $\delta(G) \geq 3$ . We will also use the following lemma to ensure that if  $\delta(G) \geq 3$  then any random endpoint of a random edge of the graph is part of any feedback vertex set of  $G$  with probability at least  $\frac{1}{4}$ .

► **Lemma 2** ([8]). *Let  $G$  be a multigraph on  $n$  vertices, with minimum degree at least 3. Then, for every feedback vertex set  $X$  of  $G$ , at least half of the edges of  $G$  have at least one endpoint in  $X$ .*

### 3 Algorithm 1

In this section, we present the first randomized  $(1 + \epsilon)$  approximation algorithm for FVS for every  $\epsilon \in (0, 1)$ . Given an instance  $(G, k)$  of FVS, we first apply reduction rules 2-4 on the graph. After applying the reduction rules, we pick all the vertices having a self-loop into the set  $S_1$  and we decrease the parameter by the number of vertices picked into  $S_1$ . If no vertex has a self-loop, then we pick an edge  $uv$  uniformly at random and add both  $u$  and  $v$  into  $S_1$  if none of the reduction rules are applicable and  $G - S_1$  is not acyclic. We decrease the parameter by 1 in this case. We do that because with good probability that one of these vertices belongs to any feedback vertex set of the graph, and hence we decrease  $\text{fvs}(G)$  by one with good probability. Then we delete  $S_1$  from the graph and repeat the same process until  $G - S_1$  becomes acyclic or the parameter decreases by at least  $\epsilon k$ . Next, we check whether  $G - S_1$  is acyclic and  $|S_1| \leq (1 + \epsilon)k$ . If yes, then we just return  $S_1$  as a solution. Otherwise, we apply the randomized FPT algorithm of [16] for FVS on the graph  $G - S_1$  with the remaining parameter. If the randomized FPT algorithm of [16] returns a solution  $S_2$ , then we return  $S_1 \cup S_2$  as a solution, otherwise, we return No. We describe the algorithm formally in Algorithm 1. Now we state the main result of this section.

► **Theorem 3.** *There exists a randomized algorithm running in  $\mathcal{O}^*(2^{\epsilon k} \cdot 2.7^{(1-\epsilon)k})$  time such that, given an FVS instance  $(G, k)$  and  $\epsilon \in (0, 1)$ , it either reports a failure or finds a feedback vertex set of  $G$  of size at most  $(1 + \epsilon)k$ . Moreover, if the algorithm is given a Yes-instance, it returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $1/e$ .*

Let us call the algorithm of Theorem 3 RPAS1. It is obtained by repeating Algorithm 1 multiple times to get a constant success probability. So before we give the proof of Theorem 3, we prove a couple of lemmas about Algorithm 1.

► **Lemma 4.** *If Algorithm 1 returns a set  $S$  then  $S$  is a feedback vertex set in  $G$  of size at most  $(1 + \epsilon)k$ .*

**Proof.** The returned solution is either of the form  $S_1$  or of the form  $S_1 \cup S_2$  for some vertex sets  $S_1$  and  $S_2$ . If it consists of only  $S_1$ , then by line 15 of the algorithm it is clear that  $G - S_1$  is acyclic and  $|S_1| \leq (1 + \epsilon)k$ .

Now, if the returned solution is of the form  $S_1 \cup S_2$ , then by the correctness of the randomized FPT algorithm of [16] for FVS, we can say that  $S_2$  is a feedback vertex set of  $G - S_1$ , and this implies that  $S = S_1 \cup S_2$  is a feedback vertex set of  $G$ . Also, if the parameter is decreased by  $\beta$  inside the while loop when Algorithm 1 returns a solution of the form  $S_1 \cup S_2$ , then observe that  $|S_1| \leq \beta + \epsilon k$  and  $|S_2| \leq k - \beta$ . Thus, we get  $|S| = |S_1| + |S_2| \leq \beta + \epsilon k + k - \beta = (1 + \epsilon)k$ . ◀

■ **Algorithm 1** First randomized  $(1 + \epsilon)$  approximation algorithm for FVS.

---

**Input** : An instance  $(G, k)$  of FVS and a  $\epsilon \in (0, 1)$ .

- 1 *Fix*  $l' = k$ ;
- 2 *Fix*  $l = (1 - \epsilon)k$ ;
- 3 *Initialize*,  $S_1 \leftarrow \emptyset$ ;
- 4 **while**  $k > l$  &  $G - S_1$  is not acyclic **do**
- 5     apply reduction rules 2-4 exhaustively to  $G - S_1$ ;
- 6     **if** there is a self-loop **then**
- 7          $S_1 = S_1 \cup \{v : \text{there is a loop on } v\}$ ;
- 8          $k \leftarrow k - |\{v : \text{there is a loop on } v\}|$ ;
- 9     **else**
- 10         pick an edge  $e = uv$  uniformly at random from  $E(G - S_1)$ ;
- 11          $S_1 = S_1 \cup \{u, v\}$ ;
- 12          $k \leftarrow k - 1$ ;
- 13     **end**
- 14 **end**
- 15 **if**  $G - S_1$  is acyclic &  $|S_1| \leq (1 + \epsilon)l'$  **then**
- 16     return  $S_1$ ;
- 17 **end**
- 18 apply the randomized FPT algorithm of [16] for FVS on  $(G - S_1, k)$ ;
- 19 **if** Above algorithm returns a solution  $S_2$  **then**
- 20     return  $S_1 \cup S_2$ ;
- 21 **else**
- 22     return No;
- 23 **end**

---

► **Lemma 5.** *Given an Yes-instance of FVS, Algorithm 1 returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $c \cdot 2^{-\epsilon k}$  for some constant  $c \geq \frac{1}{2}$ .*

**Proof.** Let  $(G, k)$  be a given Yes-instance for FVS. Notice that, inside the while loop, the parameter decreases when we pick vertices having a self-loop on them or when we choose an edge uniformly at random and add both of its endpoints to  $S_1$ . We do the latter only if none of the reduction rules are applicable.

Now, when we pick vertices having a self-loop into the set  $S_1$ , by the correctness of reduction rule 1, there must exist a feedback vertex set  $F$  of  $G$  of size at most  $k$  (as the given instance is a Yes-instance) containing those vertices having self-loops. Then we choose an edge  $uv$  uniformly at random and add both its endpoints  $u$  and  $v$  to  $S_1$  only if none of the reduction rules are applicable and thus the minimum degree of the graph is at least 3. Then by Lemma 2, at least one of  $u$  and  $v$  is in  $F$  with probability at least  $\frac{1}{2}$ . Thus, when the parameter decreases by one, we add at least one vertex of  $F$  to  $S_1$  with probability at least  $\frac{1}{2}$  (the statement is true with probability 1 when we are adding the vertices having self-loops to  $S_1$ ). If the parameter decreases by  $\beta$  (note that  $\beta \geq \epsilon k$ ) inside the while loop, then  $S_1$  contains at least  $\beta$  vertices of  $F$  with probability at least  $\frac{1}{2^{\epsilon k}}$  (as we choose an edge uniformly at random inside the while loop for at most  $\epsilon k$  steps). Also, if  $S_1$  contains at least  $\beta$  vertices from  $F$ , then  $(G - S_1, k - \beta)$  is a Yes-instance and the randomized FPT algorithm of [16] will find feedback vertex set in  $G - S_1$  of size at most  $k - \beta$  with probability at least  $c$  for some constant  $c \geq \frac{1}{2}$ . Thus, given a Yes-instance  $(G, k)$  of FVS, Algorithm 1 returns a feedback vertex set of size at most  $(1 + \epsilon)k$  with probability at least  $c \cdot 2^{-\epsilon k}$ . ◀

■ **Algorithm 2** Second randomized  $(1 + \epsilon)$  approximation algorithm for FVS.

---

**Input** : An instance  $(G, k)$  of FVS and  $\epsilon \in (0, 1)$ .

- 1 *Initialize*,  $S \leftarrow \emptyset$ ;
- 2 **while**  $|S| < (1 + \epsilon)k$  &  $G - S$  is not acyclic **do**
- 3     apply reduction rules 2-4 exhaustively to  $G - S$ ;
- 4     **if** there is a self-loop **then**
- 5          $S = S \cup \{v : \text{there is a self-loop on } v\}$ ;
- 6     **else**
- 7         pick an edge  $e$  u.a.r from  $E(G - S)$ ;
- 8         pick a vertex  $v$  u.a.r. from the endpoints of  $e$  ;
- 9          $S = S \cup \{v\}$ ;
- 10    **end**
- 11    **if**  $G - S$  is acyclic &  $|S| \leq (1 + \epsilon)k$  **then**
- 12        return  $S$ ;
- 13    **end**
- 14 **end**
- 15 return No;

---

**Proof of Theorem 3.** The randomized FPT algorithm of [16] for FVS runs in  $\mathcal{O}^*(2.7^{k'})$  time for an instance  $(G, k')$  of FVS. Notice that, when Algorithm 1 calls the algorithm of [16], the parameter is at most  $(1 + \epsilon)k$ . Thus Algorithm 1 takes  $\mathcal{O}^*(2.7^{(1+\epsilon)k})$  time when it calls the algorithm of [16]. Except for this step, all other steps in Algorithm 1 can be done in polynomial time. So overall Algorithm 1 runs in time  $\mathcal{O}^*(2.7^{(1+\epsilon)k})$ . Also, due to Lemma 5, given a Yes-instance of FVS it outputs a solution of size at most  $(1 + \epsilon)k$  with probability at least  $c \cdot 2^{-\epsilon k}$ . The algorithm of Theorem 3 (which we call RPAS1) repeats Algorithm 1 at most  $\frac{1}{c} \cdot 2^{\epsilon k}$  times. The first time Algorithm 1 returns a solution, RPAS1 returns the same solution and stops. Otherwise, if all the  $\frac{1}{c} \cdot 2^{\epsilon k}$  runs of Algorithm 1 return No, then RPAS1 returns No as well. This gives the running time of RPAS1 to be  $\mathcal{O}^*(2^{\epsilon k} \cdot 2.7^{(1+\epsilon)k})$ . We have already seen in Lemma 4 that Algorithm 1 either fails or returns a solution of size at most  $(1 + \epsilon)k$ , so this is true for RPAS1 as well. For showing the second part of Theorem 3, we observe that RPAS1 fails if and only if each of the runs of Algorithm 1 fails. Hence, RPAS1 succeeds with probability at least  $1 - (1 - c \cdot 2^{-\epsilon k})^{\frac{1}{c} \cdot 2^{\epsilon k}} \geq 1/e$ . This finishes the proof of the theorem. ◀

## 4 Algorithm II

In this section, we present our second randomized  $(1 + \epsilon)$  approximation algorithm for FVS for every  $\epsilon \in (0, 1)$ . Given an instance  $(G, k)$  of FVS, like before, we first apply reduction rules 2-4 on the graph. When none of the reduction rules 2-4 are applicable, we pick all the vertices with self-loops into the set  $S$ . If no vertex has a self-loop, then we pick an edge uniformly at random and then we pick an endpoint (say  $v$ ) of this edge uniformly at random. We add this vertex  $v$  into  $S$ . Then we delete  $S$  from the graph and repeat the same process until  $G - S$  becomes acyclic or the size of  $S$  crosses  $(1 + \epsilon)k$ . If  $G - S$  becomes acyclic before the size of  $S$  crosses  $(1 + \epsilon)k$ , then we return  $S$  as a solution, otherwise, we return No. We describe the algorithm formally in Algorithm 2. Now we state the main result of this section.

► **Theorem 6.** *There exists a randomized algorithm running in  $\mathcal{O}^*\left(\left(\left(\frac{4}{1+\epsilon}\right)^{(1+\epsilon)} \cdot \left(\frac{\epsilon}{3}\right)^\epsilon\right)^k\right)$  time such that, given an FVS instance  $(G, k)$  and  $\epsilon \in (0, 1)$ , it either reports a failure or finds a feedback vertex set of  $G$  of size at most  $(1 + \epsilon)k$ . Moreover, if the algorithm is given a Yes-instance, it returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $1/e$ .*

Let us call the algorithm of Theorem 6 RPAS2. Just like RPAS1, it is obtained by repeating Algorithm 2 multiple times to get a constant success probability. So before we give the proof of Theorem 6, we need to show some results about the success probability of Algorithm 2. We first make the following observation which follows directly from line 11 of the algorithm.

► **Observation 7.** *If Algorithm 2 returns a solution  $S$  for a given instance  $(G, k)$  of FVS then  $S$  is a feedback vertex set in  $G$  of size at most  $(1 + \epsilon)k$ .*

Let  $T(b, k, n) : \mathbb{Z} \times \mathbb{N} \times \mathbb{N} \rightarrow [0, 1]$  be defined as follows.

$$T(b, k, n) := \min_{G: |V(G)| \leq n, \text{fvs}(G) \leq k} \{\rho \mid \text{Algorithm 2 returns a feedback vertex set of } G \text{ of size at most } b \text{ with probability } \rho\}$$

If we can show a lower bound on  $T(b, k, n)$  where  $b = (1 + \epsilon)k$ , and  $k$  and  $\epsilon$  are as in the input of Algorithm 2, then that will give a lower bound for the success probability of Algorithm 2. We first make the following observation.

► **Observation 8.**  *$T(b, k, n) \leq T(b, k, n - 1)$  and  $T(b, k, n) \leq T(b, k - 1, n)$ .*

**Proof.** The first part of the observation follows from the fact that the set of graphs considered for  $T(b, k, n - 1)$  while taking the minimum in the definition is a subset of the set of graphs considered for  $T(b, k, n)$ . The second part also follows by a similar reasoning on the size of the feedback vertex set. ◀

Now, let us see what happens when Algorithm 2 adds  $v_i$  to  $S$ . Let  $E_1$  denote the event when this  $v_i$  belongs to some feedback vertex set of  $G$  of size at most  $k$ , and let  $E_2$  denote the event when  $v_i$  is not part of any feedback vertex set of  $G$  of size at most  $k$ . The probability of success of Algorithm 2 on  $G$  with parameter  $k$  and budget  $b$  is at least  $T(b - 1, k - 1, n - 1)$  and  $T(b - 1, k, n - 1)$  respectively in case  $E_1$  or  $E_2$  happens. This gives us the following.

$$T(b, k, n) \geq \Pr[E_1] \cdot T(b - 1, k - 1, n - 1) + \Pr[E_2] \cdot T(b - 1, k, n - 1).$$

Now, if we add  $v_i$  to  $S$  from line 5 of Algorithm 2 and  $G$  has a feedback vertex set of size at most  $k$ , then there must exist a feedback vertex set  $F$  of  $G$  of size at most  $k$  that contains  $v_i$ . Else, if  $v_i$  is added to  $S$  from line 9, then none of the reduction rules are applicable and by Lemma 2, we have that,  $\Pr[E_1] \geq \frac{1}{4}$ . That is,  $\Pr[E_1] = \frac{1}{4} + c_i$  for some  $c_i \geq 0$ . We also know that  $\Pr[E_1] + \Pr[E_2] = 1$ . Hence, the recurrence relation of success probability is

$$T(b, k, n) \geq \left(\frac{1}{4} + c_i\right) \cdot T(b - 1, k - 1, n - 1) + \left(\frac{3}{4} - c_i\right) \cdot T(b - 1, k, n - 1).$$

Now, using Observation 8, we can write

$$T(b, k, n) \geq \frac{1}{4} \cdot T(b - 1, k - 1, n) + \frac{3}{4} \cdot T(b - 1, k, n). \quad (1)$$

Since  $n$  is an invariant in the above recurrence relation, we can rewrite 1 as

$$T(b, k) \geq \frac{1}{4} \cdot T(b-1, k-1) + \frac{3}{4} \cdot T(b-1, k). \quad (2)$$

It can be easily seen that two trivial base cases of the recurrence 2 are the following.

i)  $T(b, k) = 0$  when  $b < 0$ , and ii)  $T(b, k) = 1$  when  $k = 0$  and  $b \geq 0$ .

#### 4.1 Solution of recurrence relation 2 and proof of Theorem 6

To solve the recurrence 2, we make use of the results in [15]. Let, a recurrence relation define a function  $p : \mathbb{Z} \times \mathbb{N} \rightarrow [0, 1]$  satisfying the following equations.

$$\begin{aligned} p(b, k) &= \min_{\{1 \leq j \leq N \mid \bar{k}^j \leq k\}} \sum_{i=1}^{r_j} \bar{\gamma}_i^j \cdot p\left(b - \bar{b}_i^j, k - \bar{k}_i^j\right) \\ p(b, k) &= 0 & \forall b < 0, k \in \mathbb{N} \\ p(b, 0) &= 1 & \forall b \geq 0, \end{aligned} \quad (3)$$

where  $N \in \mathbb{N}$ , and for any  $1 \leq j \leq N$  the following hold:  $\bar{b}^j \in \mathbb{N}_+^{r_j}$ ,  $\bar{k}^j \in \mathbb{N}^{r_j}$  and  $\bar{\gamma}^j \in \mathbb{R}_+^{r_j}$  with  $\sum_{i=1}^{r_j} \bar{\gamma}_i^j = 1$ . We say that  $\bar{k}^j \leq k$  if  $\bar{k}_i^j \leq k, \forall 1 \leq i \leq r_j$ . We refer to the recurrence relation in 3 as the composite recurrence of  $\{(\bar{b}^j, \bar{k}^j, \bar{\gamma}^j) \mid 1 \leq j \leq N\}$ . Note that for the recurrence to be properly defined, there must be  $1 \leq j \leq N$  such that  $\bar{k}^j \leq 1$  (otherwise the min operation in 3 may be taken over an empty set). Also notice that the recurrence 2 is a composite recurrence with  $N = 1$  (and thus  $j = 1$ ),  $r_j = r = 2$ ,  $\bar{\gamma}_1 = \frac{1}{4}$ ,  $\bar{\gamma}_2 = \frac{3}{4}$ ,  $\bar{b}_1 = 1$ ,  $\bar{b}_2 = 1$ ,  $\bar{k}_1 = 1$  and  $\bar{k}_2 = 0$ . Throughout this subsection, we use the word term when referring to triples of the form  $(\bar{b}^j, \bar{k}^j, \bar{\gamma}^j)$ .

We say that a vector  $\bar{q} \in \mathbb{R}_{\geq 0}^r$  is a distribution if  $\sum_{i=1}^r \bar{q}_i = 1$  and use  $D(\cdot \parallel \cdot)$  to denote **Kullback-Leibler divergence** [7]<sup>2 3</sup>.

To state the main result of [15], we need the next definition. For short, associate the term  $(\bar{b}, \bar{k}, \bar{\gamma})$  with the expression  $\sum_{i=1}^r \bar{\gamma}_i \cdot p(b - \bar{b}_i, k - \bar{k}_i)$ .

► **Definition 9.** Let  $\bar{b} \in \mathbb{N}_+^r$ ,  $\bar{k} \in \mathbb{N}^r$  and  $\bar{\gamma} \in \mathbb{R}_{\geq 0}^r$  with  $\sum_{i=1}^r \bar{\gamma}_i = 1$ . Then for  $\alpha > 0$ , the  $\alpha$  branching number of the term  $(\bar{b}, \bar{k}, \bar{\gamma})$  is the optimal value  $M^*$  of the following minimization problem over  $\bar{q} \in \mathbb{R}_{\geq 0}^r$  :

$$M^* = \min \left\{ \frac{1}{\sum_{i=1}^r \bar{q}_i \cdot \bar{k}_i} D(\bar{q} \parallel \bar{\gamma}) \mid \sum_{i=1}^r \bar{q}_i \cdot \bar{b}_i \leq \alpha \sum_{i=1}^r \bar{q}_i \cdot \bar{k}_i, \bar{q} \text{ is a distribution} \right\}.$$

If the optimization above does not have a feasible solution then  $M^* = \infty$ .

The main result of [15] is the following.

► **Theorem 10** ([15]). Let  $p$  be the composite recurrence of  $\{(\bar{b}^j, \bar{k}^j, \bar{\gamma}^j) \mid 1 \leq j \leq N\}$ , and  $\alpha > 0$ . Denote by  $M_j$  the  $\alpha$ -branching number of  $(\bar{b}^j, \bar{k}^j, \bar{\gamma}^j)$ , and let  $M = \max \{M_j \mid 1 \leq j \leq N\}$ . If  $M < \infty$  then

$$\lim_{k \rightarrow \infty} \frac{\log p(\lfloor \alpha k \rfloor, k)}{k} = -M.$$

<sup>2</sup> Formally, for  $\bar{c}, \bar{d} \in \mathbb{R}^k$  define  $D(\bar{c} \parallel \bar{d}) = \sum_{i=1}^k \bar{c}_i \log \frac{\bar{c}_i}{\bar{d}_i}$ .

<sup>3</sup> Throughout the paper we refer by log to the natural logarithm.

## 56:10 Parameterized Approximation Scheme for Feedback Vertex Set

As said earlier, for the recurrence 2, we have  $N = 1$ , and thus,  $j$  will always have only one value, so we just ignore  $j$ . Also, we have  $r_j = r = 2$ ,  $\bar{\gamma}_1 = \frac{1}{4}$ ,  $\bar{\gamma}_2 = \frac{3}{4}$ ,  $\bar{b}_1 = 1$ ,  $\bar{b}_2 = 1$ ,  $\bar{k}_1 = 1$  and  $\bar{k}_2 = 0$ . The  $\alpha$ -branching number of  $(\bar{b}, \bar{k}, \bar{\gamma})$  is,

$$M = \min \left\{ \frac{1}{\sum_{i=1}^2 \bar{q}_i \cdot \bar{k}_i} D(\bar{q} \| \bar{\gamma}) \mid \sum_{i=1}^2 \bar{q}_i \cdot \bar{b}_i \leq \alpha \sum_{i=1}^2 \bar{q}_i \cdot \bar{k}_i, \bar{q} \text{ is a distribution} \right\}.$$

So, we have to minimize

$$\frac{1}{\sum_{i=1}^2 \bar{q}_i \cdot \bar{k}_i} D(\bar{q} \| \bar{\gamma}) \quad (4)$$

with respect to the constraint,

$$\sum_{i=1}^2 \bar{q}_i \cdot \bar{b}_i \leq \alpha \sum_{i=1}^2 \bar{q}_i \cdot \bar{k}_i \quad (5)$$

Also, since  $\bar{q}$  is a distribution, we have

$$\bar{q}_1 + \bar{q}_2 = 1 \quad (6)$$

So, using 6 in 5 we get,

$$\bar{q}_1 + \bar{q}_2 \leq \alpha \bar{q}_1 \implies \bar{q}_1 \geq \frac{1}{\alpha} \quad (7)$$

Now, from 4, we get,

$$\frac{1}{\sum_{i=1}^2 \bar{q}_i \cdot \bar{k}_i} D(\bar{q} \| \bar{\gamma}) = \frac{1}{\bar{q}_1} \left( \bar{q}_1 \log \frac{\bar{q}_1}{\bar{\gamma}_1} + \bar{q}_2 \log \frac{\bar{q}_2}{\bar{\gamma}_2} \right) = \left( \log 4\bar{q}_1 + \frac{(1 - \bar{q}_1)}{\bar{q}_1} \log \frac{4(1 - \bar{q}_1)}{3} \right).$$

Notice that, the above expression is a function of  $\bar{q}_1$  only. Let us consider the function,  $f : [\frac{1}{\alpha}, 1] \rightarrow \mathbb{R}$  such that,

$$f(x) = \left( \log 4x + \frac{1-x}{x} \log \frac{4(1-x)}{3} \right).$$

Since we want a factor  $(1 + \epsilon)$  approximation for every  $\epsilon \in (0, 1)$ , we are interested for  $1 < \alpha < 2$  only and in this range of  $\alpha$ ,  $\frac{1}{\alpha} > \frac{1}{2}$ . Now, to compute  $M$ , we need to minimize  $f(x)$ .

Here, derivative of  $f(x)$ ,

$$f'(x) = -\frac{1}{x^2} \log \frac{4(1-x)}{3} \geq 0, \forall x \in \left[ \frac{1}{4}, 1 \right].$$

This implies that the function  $f$  is a monotonically increasing function on the domain of our interest ( i.e., where  $\frac{1}{\alpha} > \frac{1}{2}$ ). Hence,  $f(x)$  is minimum at  $x = \frac{1}{\alpha}$  and this implies

$$M = f\left(\frac{1}{\alpha}\right) = \log \left( \frac{4}{\alpha} \left( \frac{4}{3\alpha} (\alpha - 1) \right)^{(\alpha-1)} \right).$$

By Theorem 10,

$$p(\alpha k, k) = \left( \frac{4}{\alpha} \left( \frac{4}{3\alpha} (\alpha - 1) \right)^{(\alpha-1)} \right)^{-k}.$$

Now, from recurrence relation 2 and putting  $\alpha = (1 + \epsilon)$ , we get,

$$T((1 + \epsilon)k, k) \geq \left( \left( \frac{4}{1 + \epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^{-k}.$$

**Proof of Theorem 6.** Showing that  $T(b, k, n) \geq \left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^{-k}$  shows that given a Yes-instance  $(G, k)$  of FVS, Algorithm 2 succeeds with probability at least  $\left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^{-k}$ . Now, the proof follows along the lines of the proof of Theorem 3, and by repeating Algorithm 2  $\left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^k$  times, we get success probability at least  $1/e$ . Since Algorithm 2 runs in polynomial time, the running time of RPAS2 is  $\mathcal{O}^* \left( \left( \left( \frac{4}{1+\epsilon} \right)^{(1+\epsilon)} \cdot \left( \frac{\epsilon}{3} \right)^\epsilon \right)^k \right)$ . ◀

## 5 Algorithm III

In this section, we present the third randomized  $(1 + \epsilon)$  approximation algorithm for FVS for every  $\epsilon \in (0, 1)$ . Given an instance  $(G, k)$  of FVS, we first apply reduction rules 2-4 on the graph. After this, we pick all the vertices with self-loops into the set  $S_1$ . If no vertex has a self-loop, then we pick an edge uniformly at random, select an endpoint  $v$  of this edge uniformly at random, and add this vertex  $v$  into  $S_1$ . Then we delete  $S_1$  from the graph and repeat the same process until  $G - S_1$  becomes acyclic or the size of  $S_1$  crosses  $(1 - \epsilon)k$ . Next, we check whether  $G - S_1$  is acyclic and  $|S_1| \leq (1 + \epsilon)k$ . If yes, then we just return  $S_1$  as a solution. Otherwise, we apply a 2-approximation algorithm [1] for FVS on the graph  $G - S_1$ . If the 2-approximate solution, say  $S_2$ , of  $G - S_1$  is of size at most  $2(k - |S_1|)$  then we return  $S_1 \cup S_2$  as a solution, otherwise we return No. We describe the algorithm formally in Algorithm 3. Now we state the main result of this section.

▶ **Theorem 11.** *There exists a randomized algorithm running in  $\mathcal{O}^*(4^{(1-\epsilon)k})$  time such that, given an FVS instance  $(G, k)$  and  $\epsilon \in (0, 1)$ , it either reports a failure or finds a feedback vertex set of  $G$  of size at most  $(1 + \epsilon)k$ . Moreover, if the algorithm is given a Yes-instance, it returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $1/e$ .*

Let us call the algorithm of Theorem 6 RPAS3. Just like RPAS1 and RPAS2, it is obtained by repeating Algorithm 3 multiple times to get a constant success probability. So before we give the proof of Theorem 11, we need to prove some lemmas about Algorithm 3.

▶ **Lemma 12.** *If Algorithm 3 returns a set  $S$  then  $S$  is a feedback vertex set in  $G$  of size at most  $(1 + \epsilon)k$ .*

**Proof.** The returned solution will be either of the form  $S_1$  or of the form  $S_1 \cup S_2$  for some vertex set  $S_1$ , and  $S_2$ . If it is only  $S_1$ , then by the algorithm, it is clear that  $G - S_1$  is acyclic and  $|S_1| \leq (1 + \epsilon)k$ .

Now, if the returned solution is of the form  $S_1 \cup S_2$ , then  $S_2$  is a 2-approximate solution for the graph  $G - S_1$ . Thus, by the correctness of 2-approximation algorithm for FEEDBACK VERTEX SET, we can say  $S_2$  is a feedback vertex set in  $G - S_1$ , and this implies,  $S = S_1 \cup S_2$  is a feedback vertex set in  $G$ .

Now from Algorithm 3, we can see,  $|S_1| \geq (1 - \epsilon)k$ . Again, Algorithm 3 returns a solution of the form  $S_1 \cup S_2$  only when  $|S_2| \leq 2(k - |S_1|)$ . Thus, we get,

$$|S| = |S_1| + |S_2| \leq |S_1| + 2(k - |S_1|) = 2k - |S_1| \leq 2k - (1 - \epsilon)k = (1 + \epsilon)k. \quad \blacktriangleleft$$

▶ **Lemma 13.** *Given an Yes-instance of FVS, Algorithm 3 returns a solution of size at most  $(1 + \epsilon)k$  with probability at least  $4^{-(1-\epsilon)k}$ .*

■ **Algorithm 3** Third randomized  $(1 + \epsilon)$ -approximation algorithm for FVS.

---

**Input** : An instance  $(G, k)$  of FVS and  $\epsilon \in (0, 1)$ .

- 1 *Initialize*,  $S_1 \leftarrow \emptyset$ ;
- 2 **while**  $|S_1| < (1 - \epsilon)k$  &  $G - S_1$  is not acyclic **do**
- 3     apply reduction rules 2-4 exhaustively to  $G - S_1$ ;
- 4     **if** there is a self-loop **then**
- 5          $S_1 = S_1 \cup \{v : \text{there is a self-loop on } v\}$ ;
- 6     **else**
- 7         pick an edge  $e$  u.a.r from  $E(G - S_1)$ ;
- 8         pick a vertex  $v$  u.a.r. from the endpoints of  $e$ ;
- 9          $S_1 = S_1 \cup \{v\}$ ;
- 10    **end**
- 11    **if**  $G - S_1$  is acyclic &  $|S_1| \leq (1 + \epsilon)k$  **then**
- 12        return  $S_1$ ;
- 13    **end**
- 14 **end**
- 15 apply 2-approximation for FVS on  $G - S_1$ ;
- 16 Let  $S_2$  be returned solution;
- 17 **if**  $|S_2| > 2(k - |S_1|)$  **then**
- 18     return No;
- 19 **else**
- 20     return  $S_1 \cup S_2$ ;
- 21 **end**

---

**Proof.** Let the given instance  $(G, k)$  be a **Yes** instance, that is, there is a feedback vertex set in  $G$  of size at most  $k$ . Let  $S_1^1$  and  $S_1^2$  be the set of vertices we add in  $S$  by the line number 5 and 9 of Algorithm 3, respectively. Notice that we add vertices to  $S_1^1$  only when there are some self-loops in the graph (i.e., reduction rule 1 is applicable). By the correctness of reduction rule 1, if  $(G, k)$  is a **Yes** instance, then there exists a feedback vertex set of size at most  $k$  containing  $S_1^1$ . Now, when we add a vertex in  $S_1^2$ , from the pseudocode of Algorithm 3, we can see none of the reduction rules are applicable and thus, we can assume the minimum degree of the graph to be at least 3. Now if we pick an edge uniformly at random and then we pick one of its endpoints,  $v$  uniformly at random into  $S_1^2$  then Lemma 2 says that,  $v$  is a part of any feedback vertex set with probability at least  $\frac{1}{4}$ . Thus, for some feedback vertex set  $F$  of size at most  $k$ ,  $pr[S_1 = S_1^1 \cup S_1^2 \subseteq F] \geq \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} \cdots \frac{1}{4} (|S_1^2| \text{ times}) = \frac{1}{4^{|S_1^2|}}$ . Since, Algorithm 3 adds at most one vertex to  $S_1^2$  at every execution of the while loop and the while loop runs for at most  $(1 - \epsilon)k$  steps, so,  $|S_1^2| \leq (1 - \epsilon)k$ . Therefore,  $pr[S_1 = S_1^1 \cup S_1^2 \subseteq F] \geq \frac{1}{4^{(1-\epsilon)k}}$ . If  $S_1 = F$ , then  $S_1$  is a feedback vertex set in  $G$  and  $|S_1| \leq k \leq (1 + \epsilon)k$  and Algorithm 3 will return a solution. Else if  $S_1 \subset F$  then  $F \setminus S_1$  is a feedback vertex set in  $G - S_1$  of size  $k - |S_1|$ . Hence, the size of the 2-approximate solution  $S_2$  will be at most  $2(k - |S_1|)$  and Algorithm 3 will return a solution  $S = S_1 \cup S_2$ . So, if the given instance  $(G, k)$  is a yes-instance then for some feedback vertex set  $F$  in  $G$  of size at most  $k$ ,  $S_1 \subseteq F$  with probability at least  $\frac{1}{4^{(1-\epsilon)k}}$  and when  $S_1 \subseteq F$ , Algorithm 3 always returns a solution. Hence the proof. ◀

**Proof of Theorem 11.** There is a polynomial time 2-approximation algorithm for FVS that can be found in [1]. Also, every step under the while loop in Algorithm 3 takes only polynomial time and the while loop runs for at most  $(1 - \epsilon)k$  times. So, overall Algorithm 3

runs in polynomial time. RPAS3 calls Algorithm 3 for  $4^{(1-\epsilon)k}$  times to achieve constant success probability, and the remainder of the proof follows from Lemma 12 and Lemma 13 along the lines of proofs of Theorem 3 and Theorem 6. ◀

## 6 Comparison of RPAS1, RPAS2, and RPAS3, and proof of Theorem 1

In this section we compare the running times of the algorithms in Theorem 3, Theorem 6, and Theorem 11, which we have named RPAS1, RPAS2, and RPAS3 respectively. We observe that each of RPAS1, RPAS2 and RPAS3 performs better than the other two when  $\epsilon$  lies in the intervals  $(0, 0.176)$ ,  $(0.176, 0.674)$ , and  $(0.674, 1)$  respectively. For  $\epsilon = 0.176$ , the running times of RPAS1 and RPAS2 are the same, while for  $\epsilon = 0.674$ , the running times of RPAS2 and RPAS3 turn out to be the same. The following table gives the running time of the three algorithms for different values of  $\epsilon$  for comparison. Now we are ready to give the proof of Theorem 1.

No.	$\epsilon$	Approximation factor	RPAS1 run time	RPAS2 run time	RPAS3 run time	Better run time
1	0.05	1.05	$2.66^k$	$3.319^k$	$3.732^k$	$2.66^k$
2	0.10	1.10	$2.62^k$	$2.945^k$	$3.482^k$	$2.62^k$
3	0.15	1.15	$2.581^k$	$2.676^k$	$3.482^k$	$2.581^k$
4	0.176	1.176	$2.561^k$	$2.561^k$	$3.134^k$	$2.561^k$
5	0.20	1.20	$2.543^k$	$2.467^k$	$3.031^k$	$2.467^k$
6	0.25	1.25	$2.505^k$	$2.3^k$	$2.828^k$	$2.3^k$
7	0.30	1.30	$2.468^k$	$2.16^k$	$2.639^k$	$2.16^k$
8	0.35	1.35	$2.431^k$	$2.043^k$	$2.462^k$	$2.043^k$
9	0.40	1.40	$2.395^k$	$1.942^k$	$2.297^k$	$1.942^k$
10	0.45	1.45	$2.359^k$	$1.855^k$	$2.144^k$	$1.855^k$
11	0.50	1.50	$2.324^k$	$1.778^k$	$2^k$	$1.778^k$
12	0.55	1.55	$2.289^k$	$1.71^k$	$1.866^k$	$1.71^k$
13	0.60	1.60	$2.255^k$	$1.649^k$	$1.741^k$	$1.649^k$
14	0.65	1.65	$2.222^k$	$1.595^k$	$1.624^k$	$1.595^k$
15	0.674	1.674	$2.206^k$	$1.571^k$	$1.571^k$	$1.571^k$
16	0.70	1.70	$2.188^k$	$1.546^k$	$1.516^k$	$1.516^k$
17	0.75	1.75	$2.156^k$	$1.502^k$	$1.414^k$	$1.414^k$
18	0.80	1.80	$2.124^k$	$1.462^k$	$1.32^k$	$1.32^k$
19	0.85	1.85	$2.092^k$	$1.426^k$	$1.231^k$	$1.231^k$
20	0.9	1.90	$2.061^k$	$1.392^k$	$1.149^k$	$1.149^k$
21	0.95	1.95	$2.03^k$	$1.362^k$	$1.072^k$	$1.072^k$

**Proof of Theorem 1.** Given an instance  $(G, k)$  and an  $\epsilon \in (0, 1)$ , the algorithm runs one of RPAS1, RPAS2, and RPAS3 depending upon which of them performs the best for that value of  $\epsilon$  (for  $\epsilon = 0.176$ , we can choose either of RPAS1 or RPAS2 and for  $\epsilon = 0.674$ , we can choose either of RPAS2 or RPAS3). The running time and the correctness follows from Theorem 3, Theorem 6, and Theorem 11. ◀

### References

- 1 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.
- 2 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res.*, 12:219–234, 2000. doi:10.1613/jair.638.

- 3 Arnab Bhattacharyya, Suprovat Ghoshal, Karthik C. S., and Pasin Manurangsi. Parameterized intractability of even set and shortest vector problem from gap-eth. 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 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.17.
- 4 Ljiljana Brankovic and Henning Fernau. A novel parameterised approximation algorithm for minimum vertex cover. *Theor. Comput. Sci.*, 511:85–108, 2013. doi:10.1016/j.tcs.2012.12.003.
- 5 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 743–754. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.74.
- 6 Yijia Chen and Bingkai Lin. The constant inapproximability of the parameterized dominating set problem. *SIAM J. Comput.*, 48(2):513–533, 2019. doi:10.1137/17M1127211.
- 7 Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. URL: <http://www.elementsofinformationtheory.com/>.
- 8 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.
- 9 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.
- 10 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018. doi:10.1016/j.jcss.2017.11.001.
- 11 Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for  $k$ -cut. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2821–2837. SIAM, 2018. doi:10.1137/1.9781611975031.179.
- 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 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.
- 14 Guy Kortsarz. Fixed-parameter approximability and hardness. In *Encyclopedia of Algorithms*, pages 756–761. Springer, 2016. doi:10.1007/978-1-4939-2864-4\_763.
- 15 Ariel Kulik and Hadas Shachnai. Analysis of two-variable recurrence relations with application to parameterized approximations. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 762–773. IEEE, 2020. doi:10.1109/FOCS46700.2020.00076.
- 16 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size  $k$  in  $O^*(2.7k)$  time. *ACM Trans. Algorithms*, 18(4):34:1–34:26, 2022. doi:10.1145/3504027.
- 17 Bingkai Lin. The parameterized complexity of the  $k$ -biclique problem. *J. ACM*, 65(5):34:1–34:23, 2018. doi:10.1145/3212622.
- 18 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Fpt-approximation for FPT problems. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.
- 19 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min  $k$ -cut. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 798–809. IEEE, 2020. doi:10.1109/FOCS46700.2020.00079.

- 20 Pasin Manurangsi. Inapproximability of maximum biclique problems, minimum  $k$ -cut and densest at-least- $k$ -subgraph from the small set expansion hypothesis. *Algorithms*, 11(1):10, 2018. doi:10.3390/a11010010.
- 21 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.



# Complexity Framework for Forbidden Subgraphs III: When Problems Are Tractable on Subcubic Graphs

Matthew Johnson  

Durham University, UK

Barnaby Martin  

Durham University, UK

Sukanya Pandey  

Utrecht University, The Netherlands

Daniël Paulusma  

Durham University, UK

Siani Smith  

University of Bristol, UK

Heilbronn Institute for Mathematical Research, Bristol, UK

Erik Jan van Leeuwen  

Utrecht University, The Netherlands

---

## Abstract

For any finite set  $\mathcal{H} = \{H_1, \dots, H_p\}$  of graphs, a graph is  $\mathcal{H}$ -subgraph-free if it does not contain any of  $H_1, \dots, H_p$  as a subgraph. In recent work, meta-classifications have been studied: these show that if graph problems satisfy certain prescribed conditions, their complexity can be classified on classes of  $\mathcal{H}$ -subgraph-free graphs. We continue this work and focus on problems that have polynomial-time solutions on classes that have bounded treewidth or maximum degree at most 3 and examine their complexity on  $H$ -subgraph-free graph classes where  $H$  is a connected graph. With this approach, we obtain comprehensive classifications for (INDEPENDENT) FEEDBACK VERTEX SET, CONNECTED VERTEX COVER, COLOURING and MATCHING CUT. This resolves a number of open problems.

We highlight that, to establish that INDEPENDENT FEEDBACK VERTEX SET belongs to this collection of problems, we first show that it can be solved in polynomial time on graphs of maximum degree 3. We demonstrate that, with the exception of the complete graph on four vertices, each graph in this class has a minimum size feedback vertex set that is also an independent set.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Graph theory; Theory of computation  $\rightarrow$  Graph algorithms analysis; Theory of computation  $\rightarrow$  Problems, reductions and completeness

**Keywords and phrases** forbidden subgraphs, independent feedback vertex set, treewidth

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.57

**Related Version** *Related Version:* <https://arxiv.org/abs/2305.01104>

**Acknowledgements** We are grateful to Jelle Oostveen and Hans Bodlaender for useful discussions.

## 1 Introduction

A graph  $G$  contains a graph  $H$  as a *subgraph* if  $H$  can be obtained from  $G$  by vertex deletions and edge deletions; else  $G$  is said to be  *$H$ -subgraph-free*. If  $H$  can be obtained from  $G$  using *only* vertex deletions, then  $H$  is an *induced* subgraph of  $G$ , and if not then  $G$  is  *$H$ -free*. There are few studies of complexity classifications of graph problems for  $H$ -subgraph-free graphs (compare the greater attention given to problems on  $H$ -free graphs). There are results for INDEPENDENT SET, DOMINATING SET and LONGEST PATH [1], Max-Cut [13] and LIST COLOURING [10] (see also [11] for a shorter alternative proof of the result for INDEPENDENT SET). In these papers, complete classifications are presented giving the complexity of the problems even for  $\mathcal{H}$ -subgraph-free graphs, where  $\mathcal{H}$  is any finite set of graphs (for a set of



© Matthew Johnson, Barnaby Martin, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 57; pp. 57:1–57:15

Leibniz International Proceedings in Informatics



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

graphs  $\mathcal{H}$ , a graph  $G$  is  $\mathcal{H}$ -subgraph-free if  $G$  is  $H$ -subgraph-free for every  $H \in \mathcal{H}$ ). Such classifications seem difficult to obtain. For example, for COLOURING, there is only a partial classification [11]. For this reason – and also noting that the classifications for the problems above were all the same – a systematic approach was developed in [12] with the introduction of a new framework which we will describe after introducing some terminology.

For an integer  $k \geq 1$ , the  $k$ -subdivision of an edge  $e = uv$  of a graph replaces  $e$  by a path of length  $k + 1$  with endpoints  $u$  and  $v$  (and  $k$  new vertices). The  $k$ -subdivision of a graph  $G$  is the graph obtained from  $G$  after  $k$ -subdividing each edge. For a graph class  $\mathcal{G}$  and an integer  $k$ , let  $\mathcal{G}^k$  consist of the  $k$ -subdivisions of the graphs in  $\mathcal{G}$ . Let  $\Pi$  be a graph problem. We say that  $\Pi$  is NP-complete under edge subdivision of subcubic graphs if there exists an integer  $k \geq 1$  such that the following holds for the class of subcubic graphs  $\mathcal{G}$ : if  $\Pi$  is NP-complete for  $\mathcal{G}$ , then  $\Pi$  is NP-complete for  $\mathcal{G}^{kp}$  for every integer  $p \geq 1$ . A graph problem  $\Pi$  is a C123-problem (belongs to the framework) if it satisfies the three conditions:

- C1.  $\Pi$  is polynomial-time solvable for every graph class of bounded treewidth;
- C2.  $\Pi$  is NP-complete for the class of subcubic graphs; and
- C3.  $\Pi$  is NP-complete under edge subdivision of subcubic graphs.

As shown in [12], C123-problems allow for full complexity classifications for  $\mathcal{H}$ -subgraph-free graphs (as long as  $\mathcal{H}$  has finite size). A subdivided claw is a graph obtained from a claw (4-vertex star) after subdividing, separately, each of its edges zero or more times. The disjoint union of two vertex-disjoint graphs  $G_1$  and  $G_2$  has vertex set  $V(G_1) \cup V(G_2)$  and edge set  $E(G_1) \cup E(G_2)$ . The set  $\mathcal{S}$  consists of the graphs that are disjoint unions of subdivided claws and paths. Now, let  $\Pi$  be a C123-problem. For a finite set  $\mathcal{H}$ , the problem  $\Pi$  on  $\mathcal{H}$ -subgraph-free graphs is polynomial-time solvable if  $\mathcal{H}$  contains a graph from  $\mathcal{S}$  and NP-complete otherwise. [12].

Examples of C123-problems include INDEPENDENT SET, DOMINATING SET, LIST COLOURING, ODD CYCLE TRANSVERSAL, MAX CUT and STEINER TREE; see [12] for a comprehensive list. Thus we see the power of the framework to aid progress in deciding the complexity of problems on  $\mathcal{H}$ -subgraph-free graphs. But there are still many graph problems that are not C123. In [15], results were obtained for problems that satisfy C1 and C2 but not C3. Such problems are called C12-problems and include  $k$ -INDUCED DISJOINT PATHS,  $C_5$ -COLOURING, HAMILTON CYCLE and STAR 3-COLOURING [15]. We consider the research question:

*How do C13-problems – that is, problems that satisfy C1 and C3 but not C2 – behave for  $\mathcal{H}$ -subgraph-free graphs? Can we still classify their computational complexity?*

Let us immediately note some redundancy in the definition of C13-problems: if a problem does not satisfy C2, then C3 is implied. Nevertheless we retain the terminology to preserve the link to the approach of [12]. To show a problem is a C13 problem there are two requirements: that the problem is polynomial-time solvable both on classes of bounded treewidth and on subcubic classes. In fact, the tractable cases for C123 problems rely on that the problems satisfy C1.

► **Theorem 1** ([12]). *Let  $\Pi$  be a problem that satisfies C1. For a finite set  $\mathcal{H}$ , the problem  $\Pi$  on  $\mathcal{H}$ -subgraph-free graphs is polynomial-time solvable if  $\mathcal{H}$  contains a graph from  $\mathcal{S}$ .*

As an important step towards a full dichotomy for C13 problems, we restrict ourselves to considering  $H$ -subgraph-free graphs where  $H$  is connected. We focus on five well-known NP-complete problems that we will see are not C123 but C13-problems: FEEDBACK VERTEX SET, INDEPENDENT FEEDBACK VERTEX SET, CONNECTED VERTEX COVER and MATCHING CUT. We introduce these problems below. With one exception, we can recognize that they are C13 problems using known results.

For a graph  $G = (V, E)$ , a set  $W \subseteq V$  is a *feedback vertex set* of  $G$  if every cycle in  $G$  contains a vertex of  $W$ . Moreover,  $W$  is an *independent feedback vertex set* if  $W$  is an independent set. We note that  $G$  has a feedback vertex set of size  $k$  if and only if the 2-subdivision of  $G$  has an independent feedback vertex set of size  $k$ . A graph  $G$  might contain no independent feedback vertex set: consider, for example, a complete graph on four or more vertices. The (INDEPENDENT) FEEDBACK VERTEX SET problem is to decide if a graph  $G$  has an (independent) feedback vertex set of size at most  $k$  for some given integer  $k$ .

A set  $W \subseteq V$  is a *connected vertex cover* of  $G$  if every edge of  $E$  is incident with a vertex of  $W$ , and moreover  $W$  induce a connected subgraph. The CONNECTED VERTEX COVER problem is to decide if a graph  $G$  has a connected vertex cover of size at most  $k$  for a given integer  $k$ . A  $k$ -colouring of  $G$  is a function  $c : V \rightarrow \{1, \dots, k\}$  such that for each edge  $uv \in E$ ,  $c(u) \neq c(v)$ . The COLOURING problem is to decide if a graph  $G$  has a  $k$ -colouring for some given integer  $k$ . A *matching cut* of a connected graph is a matching (set of pairwise non-adjacent edges) that is also an edge cut, i.e., its removal creates a disconnected graph. The MATCHING CUT problem is to decide if a connected graph has a matching cut.

## 1.1 Our Results

Both FEEDBACK VERTEX SET and INDEPENDENT FEEDBACK VERTEX SET satisfy C1 [21]. Whereas FEEDBACK VERTEX SET does have a polynomial-time algorithm on subcubic graphs [22] and thus does not satisfy C2, a polynomial-time algorithm for INDEPENDENT FEEDBACK VERTEX SET on subcubic graphs was not previously known. In Section 2, we prove the following result addressing this gap in the literature.

► **Theorem 2.** *A minimum size independent feedback vertex set of every connected subcubic graph  $G \neq K_4$  is also a minimum size feedback vertex set of  $G$ . Moreover, it is possible to find a minimum independent feedback vertex set of  $G$  in polynomial time.*

Hence, both FEEDBACK VERTEX SET and INDEPENDENT FEEDBACK VERTEX SET are C13. The other problems are also C13. Namely, CONNECTED VERTEX COVER satisfies C1 [2] and is polynomial-time solvable on subcubic graphs [22] so does not satisfy C2, while COLOURING also satisfies C1 [2] but not C2 due to Brooks' Theorem [4]. Finally, MATCHING CUT satisfies C1 [3] but not C2, due to a polynomial-time algorithm for subcubic graphs [5].

The *star*  $K_{1,s}$  is the connected graph that contains a vertex of degree  $s$  whose neighbours each have degree 1. A *subdivided star* is obtained from a star by subdividing one or more of its edges.

► **Definition 3.** *An  $S_{w,x,y,z}$  is a graph formed by subdividing each edge of a  $K_{1,4}$ ,  $w - 1$ ,  $x - 1$ ,  $y - 1$ , and  $z - 1$  times. Each of the subdivided edges is called a tentacle. The vertex of degree 4 is the centre.*

In Section 3, we investigate the structure of  $H$ -subgraph-free graphs when  $H$  is a subdivided star and use this in Section 4 to show a general approach to C13 problems that requires some additional properties (that they can be solved componentwise after, possibly, the removal of bridges). This is sufficient to obtain the following result.

► **Theorem 4.** *Let  $q$  and  $r$  be positive integers. The following problems can be solved in polynomial time on  $S_{1,1,q,r}$ -subgraph-free graphs: FEEDBACK VERTEX SET, INDEPENDENT FEEDBACK VERTEX SET, CONNECTED VERTEX COVER, COLOURING and MATCHING CUT.*

In Section 5, we obtain a hardness result.

► **Theorem 5.** *FEEDBACK VERTEX SET and INDEPENDENT FEEDBACK VERTEX SET are NP-complete on the class of  $S_{2,2,2,2}$ -subgraph free graphs that have maximum degree 4.*

## 1.2 State-of-the-Art Summaries

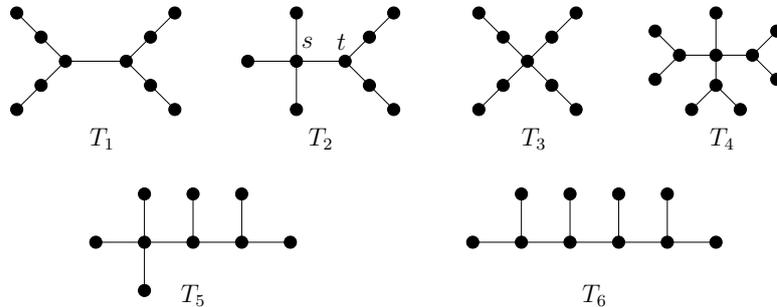
We now state complexity classifications for each of the problems. These results, proved in Section 6, combine the results above with a number of other results from [5, 7, 8, 9, 11, 14, 17, 19, 20]. None of these papers presented general results for C13 problems. However, we note, for example, that hardness when  $H$  contains a cycle follows from past results on classes of bounded girth which were proved separately for each problem, but often using a similar technique. There are other results that just apply to one or two of the problems.

► **Theorem 6.** *Let  $H$  be a connected graph. On  $H$ -subgraph-free graphs, FEEDBACK VERTEX SET and INDEPENDENT FEEDBACK VERTEX SET are solvable in polynomial time if  $H \in \mathcal{S} \cup \{S_{1,1,q,r} \mid q \geq r \geq 1\}$ . They are NP-complete if  $H$  contains a cycle or more than one vertex of degree at least 3 or  $H \in \{K_{1,5}, S_{2,2,2,2}\}$ .*

► **Theorem 7.** *Let  $H$  be a connected graph. On  $H$ -subgraph-free graphs, CONNECTED VERTEX COVER is solvable in polynomial time if  $H \in \mathcal{S} \cup \{S_{1,1,q,r} \mid q \geq r \geq 1\}$ . It is NP-complete if  $H$  contains a cycle or  $H = K_{1,5}$ .*

The following result refers to trees defined in Figure 1.

► **Theorem 8.** *Let  $H$  be a connected graph. On  $H$ -subgraph-free graphs, COLOURING is solvable in polynomial time if  $H \in \mathcal{S} \cup \{S_{1,1,q,r} \mid q \geq r \geq 1\}$  or if  $H$  is a forest with maximum degree 4 and at most seven vertices. It is NP-complete if  $H$  contains a cycle, or  $H \in \{K_{1,5}, S_{2,2,2,2}\}$ , or if  $H$  contains a subdivision of the tree  $T_1$  as a subgraph, or  $H$  contains as a subgraph the tree obtained from  $T_2$  after subdividing the edge  $st$   $p$  times,  $0 \leq p \leq 9$ , or  $H$  contains one of the trees  $S_{2,2,2,2}, T_4, T_5, T_6$  as a subgraph.*



■ **Figure 1** Illustration of the trees  $T_1, \dots, T_6$  reproduced from [11]; note that  $T_3 = S_{2,2,2,2}$ .

► **Theorem 9.** *Let  $H$  be a connected graph. On  $H$ -subgraph-free graphs, MATCHING CUT is solvable in polynomial time if  $H \in \mathcal{S} \cup \{S_{1,1,q,r} \mid q \geq r \geq 1\}$ . It is NP-complete if  $H$  contains a cycle or  $H = K_{1,5}$ .*

## 2 Independent Feedback Vertex Sets of Subcubic Graphs

In [22], Ueno, Kajitani and Gotoh gave a polynomial-time algorithm for FEEDBACK VERTEX SET on subcubic graphs. In this section, we prove Theorem 2 by showing that INDEPENDENT FEEDBACK VERTEX SET is also polynomial-time solvable on subcubic graphs by demonstrating that the problems are alike as, for any subcubic graph, one can find a minimum size feedback vertex set that is also an independent set (with a single exceptional case). As the problems can be solved componentwise, we consider only connected graphs.

In fact, we are going to prove a result that is an expansion of Theorem 2 that will come in handy later. We need some definitions. A *cactus* is a graph in which no two cycles have an edge in common. A cactus is *nice* if no two cycles have a vertex in common (every subcubic cactus is nice since if two cycles share a vertex but not an edge, we can find a vertex of degree at least 4). A cactus is *very nice* if every vertex belongs to exactly one cycle.

► **Theorem 10.** *Let  $G \neq K_4$  be a connected subcubic graph. Then a minimum size independent feedback vertex set of  $G$  is also a minimum size feedback vertex set of  $G$ . Moreover, there is a minimum size independent feedback vertex set of  $G$  that contains only vertices of degree 3 if and only if  $G$  is not a very nice cactus. There is a polynomial-time algorithm to find a minimum size independent feedback vertex set and if  $G$  is not a very nice cactus it finds a set that contains only vertices of degree 3.*

We observe that the theorem fails if  $G = K_4$ .

**Proof.** It will be seen that the proof implies a polynomial-time algorithm for finding an independent feedback vertex set of the size no greater than a given feedback vertex set.

A feedback vertex set of  $K_4$  must contain at least two vertices and so  $K_4$  has no independent feedback vertex set. In a very nice cactus, the minimum size of a feedback vertex set is equal to the number of cycles and one can easily find such a set that is independent if one permits the inclusion of degree 2 vertices. (For example, pick an arbitrary vertex  $v$  and form an independent feedback vertex set by taking the vertex in each cycle that is farthest from  $v$ .) If there are  $k$  (disjoint) cycles, then, considering the tree-like structure of a very nice cactus, there are  $2(k - 1)$  vertices of degree 3 that can be considered as  $k - 1$  adjacent pairs. Thus no set of  $k$  vertices of degree 3 is independent.

So suppose that  $G \neq K_4$  is not a very nice cactus. Of course, we may also assume that  $G$  is not a tree, else the empty set is an independent feedback vertex set. Let  $F$  be a feedback vertex set of  $G$ . To prove the theorem, we must show that we can find an independent feedback vertex set of  $G$  that is no larger than  $F$ . We can assume that  $F$  contains only vertices of degree 3 since any vertex of degree 2 can be replaced by a nearest vertex of degree 3. As  $G$  is neither a tree nor a cycle (a cycle is a very nice cactus), we know that  $G$  has vertices of degree 3.

Let  $J = \emptyset$ . Our approach is to add vertices to  $J$  until it forms an independent feedback vertex set. We make some trivial but useful statements:

1.  $F$  is a feedback vertex set containing only vertices of degree 3,
2.  $J \subseteq F$ , and
3.  $J$  is a nonseparating independent set of  $G$ ; that is, no pair of vertices of  $J$  are joined by an edge and  $G - J$  is connected.

We will repeatedly modify  $F$  and  $J$  in such a way that these three statements remain true and the size of  $F$  does not increase and it remains a feedback vertex set. We can make the following changes without contradicting the three statements.

- We can add a vertex  $x \in F \setminus J$  to  $J$  if  $x$  has no neighbour in  $J$  and is not a cutvertex in  $G - J$ .
- If  $x \in F \setminus J$ , we can redefine  $F$  as  $(F \setminus \{x\}) \cup \{y\}$  if  $y$  is a vertex that belongs to every cycle of  $G - (F \setminus \{x\})$  and has degree 3 (that is,  $y$  belongs to every cycle of  $G$  that contains  $x$  but no other vertex of  $F$ ).

Our initial aim is to make changes so that  $G - J$  is a graph where no two cycles have a vertex in common; that is, it is a nice cactus.

▷ **Claim 11.** We can modify  $F$  and  $J$  until  $G - J$  is a nice cactus.

Proof. Assume  $G - J$  contains two cycles with a common vertex, and, therefore, as  $G$  is subcubic, a common edge, else we are done. Consider a subgraph  $K$  induced by two cycles of  $G - J$  that have a common edge (so  $K$  is 2-connected and has no cutvertex). Of course,  $F$  must contain at least one vertex of  $K$ ; let  $r$  be such a vertex.

If  $r$  has degree 3 in  $K$ , then we can add it to  $J$  since it has three neighbours in  $G - J$  (so none in  $J$ ) and is not a cutvertex in  $G - J$  since  $K - \{r\}$  is connected.

Otherwise  $r$  has degree 2 in  $K$ . Traversing edges of  $K$  away from  $r$  in either direction, let  $p$  and  $q$  be the first vertices of degree 3 in  $K$  that are reached (and  $p \neq q$  by the definition of  $K$ ). Let  $r'$  be the first vertex of degree 3 in  $G$  reached from  $r$  on the path in  $K$  towards  $p$ .

If  $r$  has a neighbour  $j \in J$ , then we can redefine  $F$  as  $F \setminus \{r\} \cup \{r'\}$  since every cycle in  $G$  containing  $r$  also contains either  $j$  or  $r'$ . Suppose instead that  $r$  has no neighbour in  $J$ . Let  $r''$  be the neighbour of  $r$  in  $G - J$  but not  $K$ . If  $r$  is not a cutvertex in  $G - J$ , then we can add  $r$  to  $J$ . If  $r$  is a cutvertex in  $G - J$ , then no cycle of  $G - J$  includes the edge  $rr''$ . Thus, again, we can redefine  $F$  as  $F \setminus \{r\} \cup \{r'\}$ .

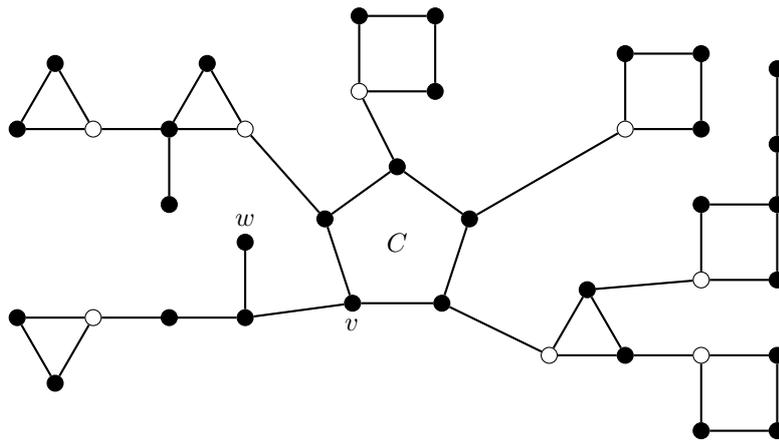
So we either add a vertex to  $J$  or modify  $F$  by replacing a vertex with another that is closer in  $K$  to  $p$ . By repetition, we either add a vertex to  $J$  or modify  $F$  to include  $p$  in which case, as noted above, we can add  $p$  to  $J$ . Therefore, if  $G - J$  contains two cycles with a common edge, we can increase the size of  $J$  and so, ultimately, we can assume that  $G - J$  contains no such pair of cycles and is a nice cactus. This completes the proof of Claim 11.  $\triangleleft$

Let  $H = G - J$ . By Claim 11, the cycles of  $H$  are vertex disjoint and the graph has a treelike structure: if one replaces each cycle by a single vertex, then a tree is obtained. As  $F$  must contain at least one vertex of each cycle of  $H$ , if we add to  $J$  one vertex chosen from each cycle of  $H$  (in any way), it will be no larger than  $F$ . If we can do this in such a way that  $J$  is an independent set and each vertex has degree 3, then the proof will be complete. Thus we must describe how to choose a degree 3 vertex from each cycle of  $H$  such that the union of these vertices and  $J$  is an independent set, possibly after some further minor modifications. The reasoning about these modifications will require that  $H$  is connected so the requirement above that  $J$  be nonseparating was needed.

If  $H$  contains no cycles, then  $J$  is already an independent feedback vertex set and there is nothing to prove. Otherwise, let  $C$  be a cycle of  $H$ . Let  $S(C)$  be the set of vertices that contains, for each cycle  $C'$  of  $H$  other than  $C$ , the vertex of  $C'$  that is nearest to  $C$  in  $H$ . See Figure 2. Each vertex  $v$  of  $S(C)$  has degree 3 in  $H$  since it has two neighbours in a cycle  $C'$  and a neighbour not in  $C'$  on the path from  $v$  to  $C$ . Thus no vertex of  $S(C)$  has a neighbour in  $J$ . Moreover, clearly  $S(C)$  is an independent set. Thus  $J \cup S(C)$  is an independent set that covers every cycle of  $G$  except  $C$ . For a vertex  $v$  in  $C$ , let  $F(v) = J \cup S(C) \cup \{v\}$ . If we can find a cycle  $C$  that contains a vertex  $v$  of degree 3 not adjacent to  $J$  or to another cycle in  $H$ , then  $F(v)$  is an independent feedback vertex set and we are done.

Suppose instead that no such cycle can be found. Notice that this implies that every vertex of  $H$  belongs to a cycle. (If there was a vertex  $w$  not in a cycle, then let  $v$  be a nearest vertex to  $w$  in a cycle and then  $F(v)$  is an independent feedback vertex set of degree 3 vertices; again, see Figure 2.) So  $H$  is a very nice cactus and, by assumption,  $J \neq \emptyset$ .

Let  $j$  be a vertex in  $J$  with neighbours  $v_1, v_2$  and  $v_3$  in  $H$ . Suppose that these three vertices are in the same cycle  $C$  of  $H$ . If  $C$  is a 3-cycle, then  $\{j, v_1, v_2, v_3\}$  induces  $K_4$ , a contradiction. So we can assume that  $v_1$  and  $v_2$  are not adjacent. Then  $J_1 = J \setminus \{j\} \cup \{v_1, v_2\} \cup S(C)$  is an independent feedback vertex set of degree 3 vertices and  $|J_2| = |F|$ . Quick check: all cycles are covered by  $J_1$  since  $v_1$  and  $S(C)$  cover the cycles of  $H$  and every cycle containing  $j$  includes at least one of  $v_1$  and  $v_2$ ;  $J_1$  is independent as  $v_1$  and  $v_2$  have degree 2 in  $H$  so no other neighbour in  $J$  and are not adjacent to vertices, such as those of  $S(C)$ , that do not belong to  $C$ , and the vertices of  $S(C)$  have degree 3 in  $H$  so no neighbours in  $J$ .



■ **Figure 2** A nice subcubic cactus. The central 5-cycle is denoted  $C$  and the white vertices form the set  $S(C)$ . Note that  $w$  does not belong to any cycle and  $v$  is the nearest vertex to  $w$  in a cycle. Thus  $S(C) \cup \{v\}$  is an independent feedback vertex set for the graph.

Suppose instead that  $v_1, v_2$  and  $v_3$  do not all belong to the same cycle. Let  $C$  be the cycle that contains  $v_1$  and suppose that  $v_2$  and  $v_3$  do not belong to the same cycle as each other (one might belong to  $C$ ). Then  $J_2 = J \setminus \{j\} \cup \{v_1\} \cup S(C)$  is an independent feedback vertex set of degree 3 vertices and  $|J_2| = |F| - 1$ . Quick check: all cycles are covered by  $J_2$  since  $v_1$  and  $S(C)$  cover the cycles of  $H$  and every cycle containing  $j$  includes either  $v_1$  or both  $v_2$  and  $v_3$  and all the paths from  $v_2$  to  $v_3$  (that do not include  $j$ ) go through either a vertex of  $J$  or a vertex of  $S(C)$  as they are in different cycles in  $H$ ;  $J_2$  is independent as  $v_1$  has degree 2 in  $H$  so, as before, no other neighbour in  $J$  or  $S(C)$ , and the vertices of  $S(C)$  have degree 3 in  $H$  so no neighbours in  $J$ . ◀

### 3 Graphs Excluding Subdivided Stars as a Subgraph: Structure

Recall that the treedepth of a graph  $G$  is the minimum height of a forest  $F$  such that for every pair of adjacent vertices in  $G$ , one is the ancestor of the other in  $F$ . It is well-known that the treewidth of a graph is at most its treedepth. In this section, we aim to show that  $H$ -subgraph-free graphs, for certain  $H$ , have bounded treedepth. Then we know that problems that are tractable on classes of bounded treewidth are also tractable on these classes. Before presenting our results, we need the following result from [18].

▶ **Theorem 12** ([18]). *Let  $G$  be a graph of treedepth at least  $d$ . Then  $G$  has a subgraph isomorphic to a path of length at least  $d$ .*

Our next two theorems consider graphs  $S_{w,x,y,z}$ . By Definition 3, this graph is four paths sharing an endvertex. In a small abuse of terminology, we will use *leaf* to mean only a vertex of degree 1 that is adjacent to the centre.

▶ **Proposition 13.** *Let  $r$  be a positive integer. Then the subclass of connected  $S_{1,1,1,r}$ -subgraph-free graphs that are not subcubic has bounded treedepth.*

**Proof.** Let  $G$  be a connected  $S_{1,1,1,r}$ -subgraph-free graph that is not subcubic so contains a vertex  $v_0$  with neighbours  $v_1, v_2, v_3, v_4$ . We will show that  $G$  has treedepth at most  $2r + 2$ . Suppose instead that the treedepth of  $G$  is at least  $2r + 3$ . The graph  $G \setminus \{v_0, v_1, v_2, v_3, v_4\}$  must have treedepth at least  $2r - 2$  (since adding a vertex to a graph cannot increase the

treedepth by more than one), and therefore, by Theorem 12, it must contain a path  $P$  of length  $2r - 2$ . Let  $Q$  be a shortest path in  $G$  between  $P$  and  $v_0$  (which must exist as  $G$  is connected). Let  $z$  be the vertex where  $P$  and  $Q$  meet. Let  $P'$  be the longest subpath of  $P$  of which  $z$  is an endvertex. As  $P'$  is at least half the length of  $P$ , and  $Q$  contains at least one edge, the path  $P' \cup Q$  contains at least  $r$  edges. Thus there exists in  $G$  a subgraph isomorphic to  $S_{1,1,1,r}$ ; the centre is  $v_0$ ,  $P' \cup Q$  is the tentacle of length  $r$ , and three of  $v_1, v_2, v_3, v_4$  are the three leaves (since at most one of these four vertices can belong to  $Q$  and none belong to  $P'$ ). This contradiction completes the proof. ◀

The assumption that the graphs are connected is needed: the class of all graphs that are each a disjoint union of a path and a  $K_{1,4}$  is not subcubic but has unbounded treedepth.

Consider now the class of all connected graphs that are each obtained from a  $K_{1,4}$  after subdividing exactly one of its edges zero or more times. This is a class of graphs that are connected, not subcubic and  $S_{1,1,q,r}$ -subgraph-free for all  $r \geq q \geq 2$  and again has unbounded treedepth. Thus, in the following analogue of Proposition 13, we need an additional property. A bridge is *proper* if neither incident vertex has degree 1. A graph is *quasi-bridgeless* if it contains no proper bridge.

► **Theorem 14.** *Let  $q$  and  $r$  be positive integers. Then the subclass of connected  $S_{1,1,q,r}$ -subgraph-free graphs that are not subcubic and are quasi-bridgeless has bounded treedepth.*

**Proof.** Let  $G$  be a connected quasi-bridgeless  $S_{1,1,q,r}$ -subgraph-free graph that is not subcubic so contains a vertex  $v_0$  with neighbours  $v_1, v_2, v_3, v_4$ . We will show that  $G$  has treedepth at most  $2(q+r+3)^2 + 6$ . Suppose instead that the treedepth of  $G$  is at least  $2(q+r+3)^2 + 7$ . The graph  $J = G \setminus \{v_0, v_1, v_2, v_3, v_4\}$  must have treedepth at least  $2(q+r+3)^2 + 2$  and therefore, by Theorem 12, it must contain a path  $P$  of length  $2(q+r+3)^2 + 2$ . Let  $z$  be the middle vertex of  $P$ . We prove the following claim.

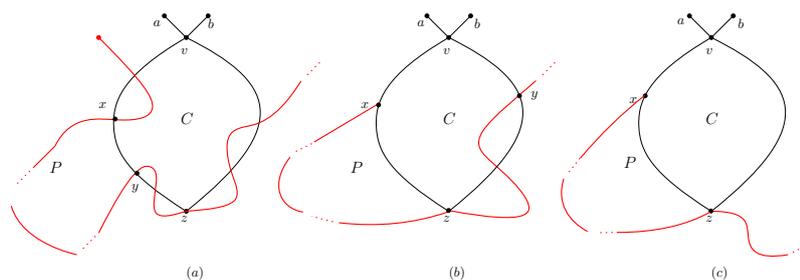
▷ **Claim 15.** If there is a cycle  $C$  in  $G$  that contains  $z$  and also a vertex  $v \neq z$  that has two neighbours  $a$  and  $b$  not on  $C$ , then  $G$  contains a subgraph isomorphic to  $S_{1,1,q,r}$ .

**Proof.** A *big adorned cycle* is a graph that contains a cycle with at least  $q+r+1$  edges and two further vertices each joined by an edge to the same vertex on the cycle; the latter vertex is called the centre. If we find a big adorned cycle in  $G$  we are done as it contains a subgraph isomorphic to  $S_{1,1,q,r}$  (the centre is the same and it is obtained by deleting one or more edges of the cycle). Let  $C^+$  be the union of  $C$  and the vertices  $a$  and  $b$  and the edges  $va$  and  $vb$ . If  $|C| \geq q+r+1$ , then  $C^+$  is a big adorned cycle.

So suppose that  $|C| \leq q+r$ . Consider the intersections of  $P$  with  $V(C^+)$ . A maximal subpath of  $P$  whose internal vertices are not in  $V(C^+)$  is called an *interval* of  $P$ . Note that  $P$  has at most  $|C^+| + 1 \leq q+r+3$  intervals. If all intervals of  $P$  have length at most  $q+r-1$ , then  $P$  itself has length at most  $(q+r+3)(q+r-1) < (q+r+3)^2$ , a contradiction. Hence, at least one of the intervals has length at least  $q+r$ ; we call such an interval *long*. See Figure 3 for an illustration.

Suppose that there is a long interval  $L$  of which both endvertices  $x$  and  $y$  are in  $V(C^+)$ . Then there are shortest (possibly trivial) paths  $S$  and  $T$  on  $C^+$  from  $v$  to  $x$  and  $y$  respectively that are vertex disjoint except for  $v$ . As  $x$  and  $y$  are distinct, the union of  $L$ ,  $S$  and  $T$  is a cycle on at least  $q+r+1$  edges. As  $v$  has four neighbours in  $C^+$ , two of them do not belong to this cycle and considering these two neighbours (and the incident edges that join them to  $v$ ) with the cycle, we have a big adorned cycle centred at  $v$ .

Hence, there is no long interval with both endvertices in  $V(C^+)$  and we can assume any long interval has just one endvertex in  $V(C^+)$ . Suppose that there are two long intervals  $L_1$  and  $L_2$  whose endvertices in  $C^+$  are  $x$  and  $y$  respectively. If  $x = y$ , then  $L_1$  and  $L_2$  are



■ **Figure 3** The cycle  $C$  and path  $P$  from the proof of Claim 15 illustrating the three cases (a) there is a long interval with both endvertices in  $P$ , (b) there are two vertex-disjoint long intervals, and (c) there are two long intervals that meet in a single vertex.

the only intervals and their union is  $P$ . This implies that  $P$  only intersects  $C^+$  in  $x$  and so we must have  $x = z$ . Then there exists in  $G$  a subgraph isomorphic to an  $S_{1,1,q,r}$  with  $z$  as its centre, the neighbours of  $z$  on  $C$  as the leaves and subpaths of  $L_1$  and  $L_2$  as the tentacles. If  $x \neq y$ , then there are shortest paths  $S, T$  on  $C^+$  from  $v$  to  $x$  and  $y$  respectively that are vertex disjoint except for  $v$ . Then there exists in  $G$  a subgraph isomorphic to an  $S_{1,1,q,r}$  with  $v$  as its centre, the paths  $S$  and  $T$ , possibly extended by subpaths of  $L_1$  and  $L_2$ , as the tentacles and two neighbours of  $v$  in  $C^+$  that do not belong to  $S$  or  $T$  as the leaves.

Hence, there is only one long interval  $L$ . As the other intervals are short, they have total length at most  $|C^+| \cdot (q+r) < (q+r+3)^2$ . Hence,  $L$  has length at least  $(q+r+3)^2 + 2$ . As  $L$  contains more than half the vertices of  $P$ , the middle vertex of  $P$  is an internal vertex of  $L$  and so does not belong to  $C^+$ . This contradicts that  $z$  is the middle vertex of  $P$  and completes the proof of the claim.  $\triangleleft$

We now apply the claim. Note that  $v_0$  and  $z$  are distinct as  $z$  belongs to  $J$  but  $v_0$  does not. Since  $G$  is quasi-bridgeless and neither  $v_0$  nor  $z$  has degree 1, it follows from Menger's Theorem [16] that there exist two edge-disjoint paths  $S, T$  from  $v_0$  to  $z$ . If  $S$  and  $T$  are internally vertex-disjoint paths, then their union forms a cycle that contains  $z$ . We can assume that each of  $S$  and  $T$  contain only one neighbour of  $v_0$  else we can find shortcuts and redefine them. Hence,  $v_0$  has two neighbours not in the cycle and we can apply Claim 15. If  $S$  and  $T$  are not internally vertex-disjoint, let  $v'$  be a vertex of  $(V(S) \cap V(T)) \setminus \{z\}$  that is furthest from  $v_0$  on  $T$ . Consider the subpath  $T'$  of  $T$  from  $v'$  to  $z$  and the subpath  $S'$  of  $S$  from  $v'$  to  $z$ . Since  $T'$  does not intersect  $S$  by definition,  $S'$  and  $T'$  are internally vertex disjoint. Hence, their union forms a cycle that contains  $z$ . Moreover,  $v'$  has degree at least four, of which two neighbours are not on  $S'$  or  $T'$ . Hence, we can apply Claim 15.  $\blacktriangleleft$

#### 4 Graphs Excluding Subdivided Stars as a Subgraph: Algorithms

We present several applications of the structural results of the previous section.

We note that FEEDBACK VERTEX SET, INDEPENDENT FEEDBACK VERTEX SET and COLOURING can be solved componentwise. In a sense, so can CONNECTED VERTEX COVER and MATCHING CUT since disconnected graphs are no instances (except possibly for CONNECTED VERTEX COVER instances with edgeless components but these can be ignored).

► **Theorem 16.** *Let  $r$  be a positive integer. A problem  $\Pi$  can be solved in polynomial time on  $S_{1,1,1,r}$ -subgraph-free graphs if the following hold:*

- i)  $\Pi$  can be solved in polynomial time on subcubic graphs,
- ii)  $\Pi$  can be solved in polynomial time on graphs of bounded treedepth, and
- iii)  $\Pi$  can be solved componentwise on disconnected graphs.

**Proof.** Let  $C$  be a connected component of a  $S_{1,1,1,r}$ -subgraph-free graph  $G$ . If  $C$  is subcubic, then the problem can be solved in polynomial time. Otherwise, by Proposition 13,  $C$  has bounded treedepth and again the problem can be solved in polynomial time. Finally, the solutions for its connected components can be merged in polynomial time. ◀

► **Theorem 17.** *Let  $q$  and  $r$  be positive integers. A problem  $\Pi$  can be solved in polynomial time on  $S_{1,1,q,r}$ -subgraph-free graphs if the following hold:*

- i)  $\Pi$  can be solved in polynomial time on subcubic graphs,
- ii)  $\Pi$  can be solved in polynomial time on graphs of bounded treedepth, and
- iii)  $\Pi$  can be solved on graphs with proper bridges using a polynomial-time reduction to a family of instances on graphs that are either of bounded treedepth or subcubic.

**Proof.** Let  $H$  be one of the family of instances obtained from an instance  $G$  of  $\Pi$ . As  $H$  is either of bounded treedepth or subcubic, the problem can be solved in polynomial time. As we have a reduction, once solved on all the family of instances, we can solve  $\Pi$  on  $G$ . ◀

The simplest way to apply Theorem 17 is to show that if it is possible to solve  $\Pi$  on each of the family of components obtained by deleting the proper bridges of an instance, then these solutions combine to provide a solution for the initial instance (since the components are quasi-bridgeless and so certainly either of bounded treedepth or subcubic by Theorem 14.)

We now use Theorem 17 to prove Theorem 4. We do not apply Theorem 16 in this paper, as the results it would give us would just be special cases of those we have obtained using Theorem 17. Nevertheless, there are potential applications of Theorem 16 as there might be C13 problems that can be solved componentwise but cannot be solved by finding the reduction required by Theorem 17. We will see, in the proof below, that to solve INDEPENDENT FEEDBACK VERTEX SET via a reduction requires an intricate argument and the careful analysis of possible solutions on subcubic graphs that was provided by Theorem 10.

► **Theorem 4 (restated).** *Let  $q$  and  $r$  be positive integers. The following problems can be solved in polynomial time on  $S_{1,1,q,r}$ -subgraph-free graphs: FEEDBACK VERTEX SET, INDEPENDENT FEEDBACK VERTEX SET, CONNECTED VERTEX COVER, COLOURING and MATCHING CUT.*

**Proof.** To show that the result follows immediately from Theorem 17, we can show that the problems can be solved by deleting bridges and considering the resulting graph componentwise; this will be trivial for some problems, but for others we will need to find a different reduction.

For FEEDBACK VERTEX SET, as bridges do not belong to cycles, the problem is unchanged when they are deleted.

For INDEPENDENT FEEDBACK VERTEX SET such a straightforward approach is not possible as if we simply delete bridges and solve the problem on the components, the merged solution might not be independent (since we might choose both endvertices of a deleted bridge). We must argue a little more carefully. Let  $G$  be a  $S_{1,1,q,r}$ -subgraph-free graph and consider the treelike structure of  $G$  when thinking of its blocks – the connected components when the bridges are deleted. In fact, consider a subgraph of  $G$  that is a block plus all its incident bridges. Some of these subgraphs might be subcubic; let us call these *C-type*. For those that are not, we can assume, by Theorem 14, that there is a constant  $c$  such that their treewidth is at most  $c$ ; let us call these subgraphs *T-type* (note that this is a weaker claim than the Theorem 14 provides, as we could assume that the treedepth was bounded). If such a subgraph is both subcubic and has treewidth at most  $c$ , we will think of it as T-type. We can assume  $c \geq 3$  so a very nice cactus (with pendant edges) is T-type. If subgraphs of

the same type overlap (because they are joined by a bridge), we observe that their union is also of that type (since the union is also either, respectively, subcubic or of treewidth at most  $c$ ). So, merging overlapping subgraphs of the same type as much as possible we can consider  $G$  as being made up of C- and T-type subgraphs and bridges that each join a C-type subgraph to a T-type subgraph. As INDEPENDENT FEEDBACK VERTEX SET can be solved in polynomial time on graphs of bounded treewidth [21] as well as on subcubic graphs by Theorem 2, we can solve it on these subgraphs. Before we solve it on a C-type subgraph, we can delete pendant bridges (that link to a T-type subgraph in  $G$ ) so the incident vertex now has degree at most 2. As a very nice cactus is being considered as a T-type subgraph, we know, by Theorem 10, that the solutions we find for C-type subgraphs do not use the vertices incident with the bridges. Thus the solutions can be merged for a solution for  $G$  that is also independent.

For CONNECTED VERTEX COVER, let  $G$  be a  $S_{1,1,q,r}$ -subgraph-free graph. Clearly, we may assume  $G$  is connected, or it has no connected vertex cover. As for INDEPENDENT FEEDBACK VERTEX SET consider each subgraph  $J$  that is a block of  $G$  and also include the bridges of  $G$  incident with the block. Observe that  $J$  is quasi-bridgeless and  $S_{1,1,q,r}$ -subgraph-free. Noticing that a connected vertex cover  $W$  of  $G$  must contain both vertices incident with any proper bridge, we see that the restriction of  $W$  to the vertices of  $J$  is a connected vertex cover of  $J$  that includes vertices incident with bridges of  $G$ . And the construction of  $J$  means its connected vertex covers will include these vertices adjacent to bridges in  $G$ . Thus we see that have a reduction and can solve the problem on  $G$ .

For COLOURING if, for a graph  $G$ , we colour the components of the graph obtained by deleting bridges, then we can merge these into a colouring of  $G$ . If the two endvertices of a bridge have been coloured alike, then we just permute the colours on one of the components. This might create new clashes, but we move to the adjacent components and permute there. By the definition of bridge, we will never have to permute the colours on a component more than once so the process terminates.

For MATCHING CUT, if a graph contains a bridge, then we have immediately that it is a yes instance. ◀

## 5 Graphs Excluding Subdivided Stars as a Subgraph: Hardness

We prove Theorem 5.

► **Theorem 5 (restated).** FEEDBACK VERTEX SET and INDEPENDENT FEEDBACK VERTEX SET are NP-complete on the class of  $S_{2,2,2,2}$ -subgraph free graphs that have maximum degree 4.

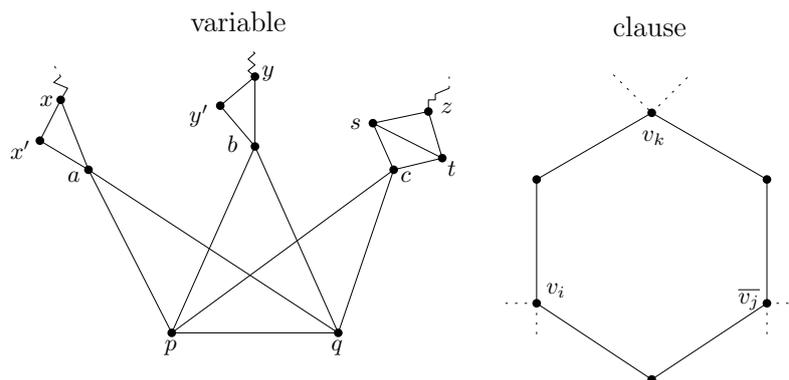
**Proof.** Both problems belong to NP. We shall show a reduction from the following NP-complete problem 2P1N-3SAT [6].

2P1N-3SAT

*Instance:* A CNF formula  $\Phi$  where each clause contains at most three literals and each variable occurs twice positively and once negatively.

*Question:* Does  $\Phi$  have a satisfying assignment?

Given an instance of 2P1N-3SAT with on variables  $\{v_1, \dots, v_n\}$ , we construct a graph  $G$  as follows. For each variable  $v_i$ , we construct the gadget shown in Figure 4. The triangles  $xx'a$  and  $yy'b$  represent the positive occurrences of the variable, while the diamond  $zstc$  represents the negative occurrence. For each clause  $C_j$ , we construct a hexagon if the clause has size 3 and a square if the clause has size 2 (we may assume that no clause has size 1).



■ **Figure 4** The variable and clause gadgets (for clauses of size 3) from the proof of Theorem 5. The vertices  $x$ ,  $y$ , and  $z$  of a variable gadget will be identified with the (labeled) vertices of clause gadgets.

Alternate vertices of this clause gadget represent literals and are identified with a vertex  $x$ ,  $y$  or  $z$  of the corresponding variable gadget. Clearly this can be done in such a way each vertex  $x$  and  $y$  of each variable gadget is identified with exactly one vertex from a clause gadget that represents a positive literal, and each vertex  $z$  of each variable gadget is identified with exactly one vertex from a clause gadget that represents a negative literal. Note that  $G$  has maximum degree 4.

▷ **Claim 18.**  $G$  does not contain  $S_{2,2,2}$  as a subgraph.

*Proof.* Let us consider where we might find the centre vertex of a  $S_{2,2,2}$  in  $G$ . Clearly a vertex  $v$  cannot be the centre vertex if its 2-neighbourhood in  $G$  contains a cut of size 3 (that is, if there are 3 vertices each of distance at most 2 from  $v$  that form a cut in  $G$ ). The centre vertex cannot be the vertices  $p$  or  $q$  of a variable gadget, because the set  $\{a, b, c\}$  of the same gadget forms a cut of size 3 in the 2-neighbourhood of  $p$  and  $q$ . The centre vertex cannot be the vertices  $a$ ,  $b$ , or  $c$  of a variable gadget either, because  $\{x, p, q\}$ ,  $\{y, p, q\}$  and  $\{z, p, q\}$  respectively form cuts of size 3 in their 2-neighbourhoods. The vertices  $x$ ,  $y$ , and  $z$  cannot be the centre vertex as in their 2-neighbourhood is a cut of size 3 that contains their two neighbours in a clause gadget and, respectively,  $a$ ,  $b$  and  $c$ . The remaining vertices of  $G$  have degree less than 4. The claim is proved. ◁

Any feedback vertex set of a variable gadget has size at least 4, because it contains four disjoint cycles. So any feedback vertex set of  $G$  must contain at least  $4n$  vertices. It only remains to show that  $G$  has an (independent) feedback vertex set of size at most  $4n$  if and only if  $\Phi$  is satisfiable.

Assume that  $\Phi$  has a satisfying assignment. We construct a feedback vertex set  $F$  of  $G$ . If a variable is true, then the vertices  $x$ ,  $y$ ,  $p$ , and  $t$  of the variable gadget belong to  $F$ . If a variable is false, then instead  $z$ ,  $a$ ,  $b$ , and  $c$  belong to  $F$ . Thus  $F$  is an independent set (vertices of distinct variable gadgets are not adjacent) and its size is exactly  $4n$ .

▷ **Claim 19.**  $F$  is a feedback vertex set.

*Proof.* Notice that if a literal of a clause is satisfied, then, in the clause gadget, the corresponding vertex is in  $F$ . Thus, as clause is satisfied, each cycle contained in a single variable or clause gadget contains a vertex of  $F$ . Consider a cycle of  $G$  that is not contained within a single gadget. It must include a non-trivial path of some variable gadget where the

endvertices are two of  $\{x, y, z\}$ . If it includes  $x$  it must also include  $a$  and if it includes  $y$  it must also include  $b$ . But  $F$  contains one of  $\{x, a\}$  and one of  $\{y, b\}$  so such a cycle also intersects  $F$ . Thus  $F$  intersects all the cycles of  $G$ .  $\triangleleft$

Conversely, suppose that  $G$  has a feedback vertex set  $F$  of size at most  $4n$ . Again, each variable gadget contains at least four vertices of  $F$  and so contains exactly four vertices of  $F$ . Notice that  $F$  cannot contain either  $\{x, z\}$  or  $\{y, z\}$  as, in each case, there are three disjoint cycles of the gadget that would need to be covered by just two vertices.

Let us describe a satisfying assignment of  $\Phi$ . If, for a variable gadget, either  $x$  or  $y$  belongs to  $F$ , we let the variable be true. If  $z$  belongs to  $F$ , we let it be false. By the preceding argument, there is no possibility that we must set a variable to be both true and false. If none of  $\{x, y, z\}$  belong to  $F$ , we set the value of the variable arbitrarily. This is a satisfying assignment as every clause gadget (which is a cycle) must have at least one vertex in  $F$  and the corresponding variable is satisfied.  $\blacktriangleleft$

## 6 Proofs of the Classifications

We prove Theorems 6–9. Noting that the theorems contain some analogous results, and wishing to avoid repetition, we make a few general comments that apply to all proofs.

We state again that the five problems under consideration are C13 problems. Thus when  $H \in \mathcal{S}$ , each theorem follows from Theorem 1. When  $H = S_{1,1,q,r}$ , we apply Theorem 4. Thus, except for Theorem 8 on COLOURING, the following proofs need only cover the NP-complete cases.

**Proof of Theorem 6.** We note again that FEEDBACK VERTEX SET reduces to INDEPENDENT FEEDBACK VERTEX SET after subdividing each edge so here we consider only the former.

By Poljak’s construction [19], for every integer  $g \geq 3$ , FEEDBACK VERTEX SET is NP-complete for graphs of girth at least  $g$  (the girth of a graph is the length of its shortest cycle). Thus FEEDBACK VERTEX SET is NP-complete for  $H$ -subgraph-free graphs whenever  $H$  contains a cycle.

Suppose that  $H$  has  $m$  vertices and more than one vertex of degree at least 3. From any graph  $G$ , if we we subdivide each edge  $m$  times, we obtain a graph  $J$  that is  $H$ -subgraph free since the distance between any pair of vertices of degree more than 2 is at least  $m + 1$ . In finding a minimum size feedback vertex set of  $J$ , we may as well restrict ourselves to selecting vertices of  $G$ . This implies that FEEDBACK VERTEX SET is NP-complete for  $H$ -subgraph-free graphs.

The problem is NP-complete on planar graphs of maximum degree 4 [20] (so for  $K_{1,5}$ -subgraph-free graphs).

Theorem 5 completes the proof.  $\blacktriangleleft$

**Proof of Theorem 7.** For every integer  $g \geq 3$ , CONNECTED VERTEX COVER is NP-complete for graphs of girth at least  $g$  [17], so also for  $H$ -subgraph-free graphs whenever  $H$  contains a cycle. It is NP-complete on graphs of maximum degree 4 [8], so for  $K_{1,5}$ -subgraph-free graphs.  $\blacktriangleleft$

**Proof of Theorem 8.** For every integer  $g \geq 3$ , COLOURING is NP-complete for graphs of girth at least  $g$  [14], so also for  $H$ -subgraph-free graphs whenever  $H$  contains a cycle. In [9], it was shown that COLOURING is NP-complete on (planar) graphs of maximum degree 4, and so too for  $K_{1,5}$ -subgraph-free graphs. The other cases are all proved in [11]  $\blacktriangleleft$

**Proof of Theorem 9.** For every integer  $g \geq 3$ , MATCHING CUT is NP-complete for graphs of girth at least  $g$  [7], so also for  $H$ -subgraph-free graphs whenever  $H$  contains a cycle. It is NP-complete on graphs of maximum degree 4 [5], so for  $K_{1,5}$ -subgraph-free graphs. ◀

## 7 Conclusions

We made significant progress towards classifying the complexity of five well-known C13-problems on  $H$ -subgraph-free graphs, extending previously known results. In particular, we identified a gap in the literature, and provided a polynomial-time algorithm for INDEPENDENT FEEDBACK VERTEX SET for subcubic graphs.

If  $H$  is connected, then we narrowed the gap for these problems to the open case where  $H = S_{1,p,q,r}$ , so  $H$  is a subdivided star with one short leg and three arbitrarily long legs. To obtain a result for connected  $S_{1,p,q,r}$ -subgraph-free graphs similar to our previous results, we would need the graphs to be 3-edge-connected. Indeed, the statement is false without this assumption. Consider the class of all graphs that are each the union of a path and a  $K_{1,4}$  two of whose leaves are identified with distinct end-vertices of the path and whose other two leaves are made adjacent. This is a class of graphs that are bridgeless, not subcubic and  $S_{1,p,q,r}$ -subgraph-free and again has unbounded treedepth. It is not yet clear whether a suitably modified theorem statement would indeed hold. In addition, it is unclear whether this would yield a result that could be applied in the same way as Proposition 13 and Theorem 14 were above. We leave the case  $H = S_{1,p,q,r}$  as future research.

Finally, we also leave determining the complexity of CONNECTED VERTEX COVER and MATCHING CUT on  $S_{2,2,2,2}$ -subgraph-free graphs as an open problem.

---

## References

- 1 Vladimir E. Alekseev and Dmitry V. Korobitsyn. Complexity of some problems on hereditary graph classes. *Diskretnaya Matematika*, 2:90–96, 1990.
- 2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
- 3 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62:109–126, 2009.
- 4 R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941. doi:10.1017/S030500410002168X.
- 5 Vašek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.
- 6 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23:864–894, 1994.
- 7 Carl Feghali, Felicia Lucke, Daniel Paulusma, and Bernard Ries. Matching cuts in graphs of high girth and  $h$ -free graphs. *CoRR*, 2212.12317, 2023. arXiv:2212.12317.
- 8 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.
- 9 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
- 10 Petr A. Golovach and Daniël Paulusma. List coloring in the absence of two subgraphs. *Discrete Applied Mathematics*, 166:123–130, 2014.
- 11 Petr A. Golovach, Daniël Paulusma, and Bernard Ries. Coloring graphs characterized by a forbidden subgraph. *Discrete Applied Mathematics*, 180:101–110, 2015.
- 12 Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Daniël Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs I: The Framework. *CoRR*, abs/2211.12887, 2022. arXiv:2211.12887.

- 13 Marcin Kamiński. Max-Cut and containment relations in graphs. *Theoretical Computer Science*, 438:89–95, 2012.
- 14 Frédéric Maffray and Myriam Preissmann. On the np-completeness of the  $k$ -colorability problem for triangle-free graphs. *Discrete Math.*, 162:313–317, 1996.
- 15 Barnaby Martin, Sukanya Pandey, Daniel Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs: When hardness is not preserved under edge subdivision. *CoRR*, 2022. [arXiv:2211.14214](https://arxiv.org/abs/2211.14214).
- 16 Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.
- 17 Andrea Munaro. Boundary classes for graph problems involving non-local properties. *Theoretical Computer Science*, 692:46–71, 2017.
- 18 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.
- 19 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 015(2):307–309, 1974. URL: <http://eudml.org/doc/16622>.
- 20 Ewald Speckenmeyer. *Untersuchungen zum Feedback Vertex Set Problem in ungerichteten Graphen*. PhD thesis, Paderborn, 1983.
- 21 Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 98-A(6):1179–1188, 2015. doi:10.1587/transfun.E98.A.1179.
- 22 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1):355–360, 1988. doi:10.1016/0012-365X(88)90226-9.



# Polynomial-Delay Enumeration of Large Maximal Common Independent Sets in Two Matroids

Yasuaki Kobayashi  

Hokkaido University, Sapporo, Japan

Kazuhiro Kurita  

Nagoya University, Nagoya, Japan

Kunihiro Wasa  

Hosei University, Tokyo, Japan

---

## Abstract

Finding a *maximum* cardinality common independent set in two matroids (also known as MATROID INTERSECTION) is a classical combinatorial optimization problem, which generalizes several well-known problems, such as finding a maximum bipartite matching, a maximum colorful forest, and an arborescence in directed graphs. Enumerating all *maximal* common independent sets in two (or more) matroids is a classical enumeration problem. In this paper, we address an “intersection” of these problems: Given two matroids and a threshold  $\tau$ , the goal is to enumerate all maximal common independent sets in the matroids with cardinality at least  $\tau$ . We show that this problem can be solved in polynomial delay and polynomial space. We also discuss how to enumerate all maximal common independent sets of two matroids in non-increasing order of their cardinalities.

**2012 ACM Subject Classification** Mathematics of computing → Enumeration

**Keywords and phrases** Polynomial-delay enumeration, Ranked Enumeration, Matroid intersection, Reverse search

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.58

**Related Version** *Full Version*: <https://arxiv.org/abs/2307.08948>

**Funding** *Yasuaki Kobayashi*: JSPS KAKENHI Grant Numbers JP20H00595 and JP23H03344  
*Kazuhiro Kurita*: JP21K17812, JP22H03549, JST CREST Grant Number JPMJCR18K3, and JST ACT-X Grant Number JPMJAX2105  
*Kunihiro Wasa*: JSPS KAKENHI Grant Numbers JP22K17849 and JP22H03549, and JST CREST Grant Number JPMJCR18K3

## 1 Introduction

The bipartite matching problem is arguably one of the most famous combinatorial optimization problems, which asks to find a maximum cardinality matching in a bipartite graph. By polynomial-time algorithms for the maximum flow problems, this problem can be solved in polynomial time. This problem is naturally generalized for non-bipartite graphs, which is also solvable in polynomial time [9].

Another natural generalization of the bipartite matching problem is MATROID INTERSECTION. In this problem, we are given two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$ , where  $\mathcal{I}_1 \subseteq 2^S$  and  $\mathcal{I}_2 \subseteq 2^S$  are the set of independent sets of  $M_1$  and  $M_2$ , respectively, and asked to find a maximum cardinality *common independent set* of  $M_1$  and  $M_2$ , that is, a maximum cardinality set in  $\mathcal{I}_1 \cap \mathcal{I}_2$ . When both  $M_1$  and  $M_2$  are partition matroids, this problem is equivalent to the bipartite  $b$ -matching problem, which is a generalization of the bipartite matching problem. A famous matroid intersection theorem [10] shows a min-max formula and also gives a polynomial-time algorithm for MATROID INTERSECTION [23].



© Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 58; pp. 58:1–58:14

Leibniz International Proceedings in Informatics



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

These classical results give efficient algorithms to find a *single* best (bipartite) matching in a graph or common independent set in two matroids. This type of objective serves as the gold standard in many algorithmic and computational studies. However, such a single best solution may not be appropriate for real-world problems due to the complex nature of them [11].

One possible remedy to this issue is to enumerate *multiple* solutions instead of a single best one. From the point of view of enumeration, the problems of enumeration *maximal* and *maximum* (bipartite) matchings and its generalization are studied in the literature [6, 29–31]. Enumerating maximal independent sets in a graph is one of the best-studied problems in this area and is solvable in polynomial delay and polynomial space [6, 29]. Due to the correspondence between matchings in a graph and independent sets in its line graph, we can enumerate all maximal matchings in polynomial delay and polynomial space as well. Moreover, several algorithms that are specialized to (bipartite) matchings are known [30, 31].

Enumeration algorithms for matroids are also frequently studied in the literature [3, 12, 16, 17, 22]. Lawler et al. [22] showed that all maximal common independent sets in  $k$  matroids can be enumerated in polynomial delay when  $k$  is constant. For general  $k$ , this problem is highly related to DUALIZATION (or equivalently, minimal transversal enumeration, minimal hitting set enumeration), which can be solved in output quasi-polynomial time<sup>1</sup> [3]. Apart from common independent sets, enumeration problems related to matroids are studied [15, 17], such as minimal multiway cuts [17] and minimal Steiner forests in graphs [15].

In this paper, we consider an “intersection” of the above two worlds, optimization and enumeration, for MATROID INTERSECTION. More specifically, given two matroids  $M_1$  and  $M_2$  and an integer  $\tau$ , we consider the problem of enumerating all *maximal* common independent sets of  $M_1$  and  $M_2$  with cardinality at least  $\tau$ . We refer to this problem as LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION. By setting  $\tau = 0$ , we can enumerate all maximal common independent sets of  $M_1$  and  $M_2$ , and by setting  $\tau = \text{opt}$ , we can enumerate all maximum common independent sets of  $M_1$  and  $M_2$ , where  $\text{opt}$  is the optimal value of MATROID INTERSECTION. We would like to mention that simultaneously handling two constraints, maximality and cardinality, would make enumeration problems more difficult (see [18–20], for other enumeration problems). We show that LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION can be solved in polynomial delay and space. This extends the results of enumerating maximum common independent sets due to [12] and enumerating maximal common independent sets due to [22]. Our enumeration algorithm allows us to enumerate several combinatorial objects with maximality and cardinality constraints, such as bipartite  $b$ -matchings, colorful forests, and degree-constraint subdigraphs.

To prove this, we devise a reverse search algorithm [1] to enumerate all maximal common independent sets of  $M_1$  and  $M_2$ . This algorithm enumerates the solutions in a depth first manner. To completely enumerate all the solutions without duplicates, we carefully design its search strategy. We exploit a famous augmenting path theorem for MATROID INTERSECTION [23]. This enables us to design a “monotone” search strategy, yielding a polynomial-delay and polynomial-space enumeration algorithm. A similar idea is used in [19] for enumerating maximal matchings with cardinality at least  $\tau$  but we need several nontrivial lemmas to obtain our result.

Although our algorithm enumerates all maximal common independent sets of two matroids with cardinality at least  $\tau$ , solutions may not be generated in a sorted order, which is of great importance in database community [8, 26]. A *ranked enumeration algorithm* is an algorithm

---

<sup>1</sup> An enumeration algorithm runs in output quasi-polynomial time if it runs in time  $N^{(\log N)^c}$ , where  $c$  is a constant and  $N$  is the combined size of the input and output.

enumerates all the solutions in a non-increasing order of their cardinality (or more generally objective value). We discuss how to convert our enumeration algorithm to the one that enumerates in a ranked manner with a small overhead in the running time.

## 2 Preliminaries

Let  $S$  be a finite set. We denote the cardinality of  $S$  as  $n$ . For two sets  $X$  and  $Y$ , the symmetric difference of  $X$  and  $Y$  is defined as  $X \triangle Y := (X \setminus Y) \cup (Y \setminus X)$ . A pair  $M = (S, \mathcal{I})$  is called a *matroid* if  $M$  satisfies the following conditions:

- $\emptyset \in \mathcal{I}$ ,
- if  $I \in \mathcal{I}$  and  $J \subseteq I$ , then  $J \in \mathcal{I}$ , and
- if  $I, J \in \mathcal{I}$  and  $|I| < |J|$ , then  $I \cup \{e\} \in \mathcal{I}$  for some  $e \in J \setminus I$ .

A subset  $S'$  of  $S$  is called an *independent set* of  $M$  (or *independent* in  $M$ ) if  $S'$  is contained in  $\mathcal{I}$  and  $S'$  is called a *dependent set* of  $M$  (or *dependent* in  $M$ ) otherwise. An inclusion-wise maximal independent set of  $M$  is called a *base* of  $M$ , and an inclusion-wise minimal dependent set of  $M$  is called a *circuit* of  $M$ . For two distinct circuits  $C_1$  and  $C_2$  of  $M$  with  $C_1 \cap C_2 \neq \emptyset$  and  $e \in C_1 \cap C_2$ , there always exists a circuit  $C_3$  of  $M$  such that  $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$ . This property is called the (*weak*) *circuit elimination axiom* [25]. For a matroid  $M = (S, \mathcal{I})$  and a subset  $X \subseteq S$ , the pair  $(X, \mathcal{J})$  is the *restriction* of  $M$  to  $X$ , where  $\mathcal{J} = \{Y \subseteq X : Y \in \mathcal{I}\}$ . We denote it as  $M \mid X$ . Similarly, the pair  $(S \setminus X, \mathcal{J}')$  is the *deletion* of  $X$  from  $M$ , where  $\mathcal{J}' = \{Y \setminus X : Y \in \mathcal{I}\}$ . We denote it as  $M \setminus X$ . Moreover, the pair  $(S \setminus X, \mathcal{J}'' )$  is the *contraction* of  $X$  from  $M$ , where  $\mathcal{J}'' = \{Y \subseteq S \setminus X : M \mid X \text{ has a base } B \text{ such that } Y \cup B \in \mathcal{I}\}$ . We denote it as  $M/X$ . Similarly, It is known that for a matroid  $M = (S, \mathcal{I})$  and  $X \subseteq S$ ,  $M/X$ ,  $M \mid X$ , and  $M \setminus X$  are all matroids [25]. For two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$  defined on the same set  $S$ , a subset  $T \subseteq S$  is a *common independent set* of  $M_1$  and  $M_2$  if  $T \in \mathcal{I}_1$  and  $T \in \mathcal{I}_2$ .

Let  $I_1$  and  $I_2$  be distinct independent sets of  $M$ . In our algorithm, we frequently consider a matroid obtained from  $M$  by restricting to  $I_1 \cup I_2$  and then contracting  $I_1 \cap I_2$ . This matroid is defined on  $I_1 \triangle I_2$  and has some properties shown below.

► **Proposition 1.** *Let  $I_1$  and  $I_2$  be independent sets of  $M$  and let  $M' = (M \mid (I_1 \cup I_2)) / (I_1 \cap I_2)$ .  $I \subseteq I_1 \triangle I_2$  is independent in  $M'$  if and only if  $I \cup (I_1 \cap I_2)$  is independent in  $M$ .*

**Proof.** Suppose that  $I$  is independent in  $M'$ . As  $M'$  is a contraction of  $I_1 \cap I_2$  from  $M \mid (I_1 \cup I_2)$ ,  $I \cup (I_1 \cap I_2)$  is independent in  $M \mid (I_1 \cup I_2)$  and hence in  $M$ . The converse direction is analogous. ◀

► **Proposition 2.** *Let  $I_1$  and  $I_2$  be maximal common independent sets of  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$ . Then, both  $I_1 \setminus I_2$  and  $I_2 \setminus I_1$  are maximal common independent sets of two matroids  $M'_1 = (M_1 \mid (I_1 \cup I_2)) / (I_1 \cap I_2)$  and  $M'_2 = (M_2 \mid (I_1 \cup I_2)) / (I_1 \cap I_2)$ .*

**Proof.** By symmetry, it suffices to show that  $I_1 \setminus I_2$  is a maximal common independent set of  $M'_1$  and  $M'_2$ . By Proposition 1,  $(I_1 \setminus I_2) \cup (I_1 \cap I_2) = I_1$  is independent in  $M_1$  if and only if  $I_1 \setminus I_2$  is independent in  $M'_1$ . Similarly,  $I_1$  is independent in  $M_2$  if and only if  $I_1 \setminus I_2$  is independent in  $M'_2$ . Thus,  $I_1 \setminus I_2$  is a common independent set of  $M'_1$  and  $M'_2$ . To see the maximality, suppose that there is  $e \in I_2 \setminus I_1$  such that  $(I_1 \setminus I_2) \cup \{e\}$  is a common independent set of  $M'_1$  and  $M'_2$ . By Proposition 1,  $I_1 \cup \{e\}$  is a common independent set of  $M_1$  and  $M_2$ , contradicting the maximality of  $I_1$ . ◀

We next define some notations for directed graphs. In this paper, we assume that directed graphs have no self-loops. For a directed graph  $D = (V, A)$ , we say that a vertex  $v$  is an *out-neighbor* of  $u$  ( $u$  is an *in-neighbor* of  $v$ ) in  $D$  if  $D$  has an arc  $(u, v)$ . The set of

out-neighbors of  $v$  is denoted by  $N^+(v)$ , and the set of in-neighbors of  $v$  is denoted by  $N^-(v)$ . A sequence  $(v_1, \dots, v_k)$  of distinct vertices is a *directed path* if there is an arc  $(v_i, v_{i+1})$  for  $1 \leq i < k$ . A directed path  $(v_1, \dots, v_k)$  in  $D$  is called a *directed path without shortcuts* if  $D$  has no arc from  $v_i$  to  $v_j$  for any  $1 \leq i < j \leq k$  with  $i + 1 < j$ .

We measure the time complexity of enumeration algorithms with delay complexity [13]. The *delay* of an enumeration algorithm is the maximum time elapsed between two consecutive outputs, including preprocessing and post-processing time. An enumeration algorithm is called a *polynomial-delay enumeration algorithm* if its delay is upper bounded by a polynomial of the size of an input. An enumeration algorithm is called a *linear incremental-time enumeration algorithm* if, for any  $i \leq N$ , an algorithm outputs at least  $i$  solutions in time  $O(i \cdot \text{poly}(n))$ , where  $N$  is the number of solutions [4].

Now, we formally define our problems. Throughout the paper, we assume that matroids are given as *independence oracles*, that is, for a matroid  $M = (S, \mathcal{I})$ , we can test whether a subset  $X \subseteq S$  belongs to  $\mathcal{I}$  by accessing an oracle for  $M$ . Moreover, we assume that independence oracles can be evaluated in  $Q$  time and in  $\hat{Q}$  space. We say that an enumeration algorithm runs in polynomial delay (resp. polynomial space) if the delay (resp. space) is upper bounded by a polynomial in  $n + Q$  (resp.  $n + \hat{Q}$ ).

► **Definition 3.** *Given two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$  represented by independence oracles and an integer  $\tau$ , LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION asks to enumerate all maximal common independent sets of  $M_1$  and  $M_2$  with cardinality at least  $\tau$ .*

► **Definition 4.** *Given two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$  represented by independence oracles, RANKED MAXIMAL COMMON INDEPENDENT SET ENUMERATION asks to enumerate all maximal common independent sets of  $M_1$  and  $M_2$  in a non-increasing order with respect to cardinality.*

## 2.1 Overview of an algorithm for finding a maximum common independent set

Our proposed algorithm for LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION leverages a well-known property used in an algorithm for finding a maximum common independent set in two matroids. In this paper, we refer to a particular algorithm given by Lawler [23].

Let  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$  be matroids. In Lawler's algorithm [23], we start with an arbitrary common independent set  $I$  of  $M_1$  and  $M_2$  (e.g.,  $I := \emptyset$ ), update  $I$  to a larger common independent set  $I'$  in some "greedy way". This update procedure is based on the following auxiliary directed graph  $D_{M_1, M_2}(I) = (S \cup \{s, t\}, A)$ .

Let  $I \subseteq S$  be a common independent set of  $M_1$  and  $M_2$ . The set  $A = A_1 \cup A_2 \cup A_3 \cup A_4$  of arcs in  $D_{M_1, M_2}(I)$  consists of the following four types of arcs. The first type of arcs is defined as

$$A_1 = \{(e, f) : e \in I, f \in S \setminus I, I \cup \{f\} \notin \mathcal{I}_1, (I \cup \{f\}) \setminus \{e\} \in \mathcal{I}_1\},$$

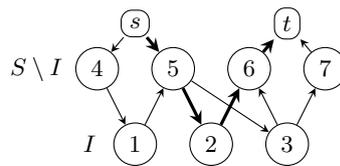
that is, an arc  $(e, f) \in A_1$  indicates that  $I \Delta \{e, f\}$  is independent in  $M_1$ . Symmetrically, the second type of arcs is defined as

$$A_2 = \{(f, e) : e \in I, f \in S \setminus I, I \cup \{f\} \notin \mathcal{I}_2, (I \cup \{f\}) \setminus \{e\} \in \mathcal{I}_2\},$$

that is, an arc  $(f, e)$  indicates that  $I \Delta \{e, f\}$  is independent in  $M_2$ . The third and fourth types of arcs are defined as

$$A_3 = \{(s, f) : f \in S \setminus I, I \cup \{f\} \in \mathcal{I}_1\}$$

$$A_4 = \{(f, t) : f \in S \setminus I, I \cup \{f\} \in \mathcal{I}_2\},$$



■ **Figure 1** This figure depicts an example of the auxiliary graph  $D_{M_1, M_2}(\{1, 2, 3\})$ . Let  $M_1$  and  $M_2$  be matroids with the same ground set  $\{1, \dots, 7\}$  that defined by five bases  $\{1, 2, 3, 4\}$ ,  $\{1, 2, 3, 5\}$ ,  $\{1, 3, 5, 6\}$ ,  $\{1, 2, 5, 6\}$ ,  $\{1, 2, 5, 7\}$  and six bases  $\{1, 2, 3, 6\}$ ,  $\{1, 2, 3, 7\}$ ,  $\{1, 2, 5, 6\}$ ,  $\{1, 3, 5, 6\}$ ,  $\{1, 2, 5, 7\}$ ,  $\{2, 3, 4, 6\}$ , respectively. In this example,  $D_{M_1, M_2}(\{1, 2, 3\})$  has a directed  $s$ - $t$  path  $P = (s, 5, 2, 6, t)$  without shortcuts and  $\{1, 3, 5, 6\}$  is a common independent set of  $M_1$  and  $M_2$ .

respectively. Arcs  $(s, f)$  and  $(f, t)$  indicate that  $I \cup \{f\}$  is independent in  $M_1$  and in  $M_2$ , respectively. We illustrate a concrete example of  $D_{M_1, M_2}(I)$  in Figure 1. In the following, we simply write  $D(I)$  to denote  $D_{M_1, M_2}(I)$ .

Let  $P$  be a directed path from  $s$  to  $t$  in  $D(I)$  without shortcuts. By the definition of  $D(I)$ ,  $|V(P) \cap I|$  is one less than  $|V(P) \setminus I|$ . Moreover, we can prove that  $I \Delta (V(P) \setminus \{s, t\})$  is a common independent set of  $M_1$  and  $M_2$  [23], meaning that the common independent set  $I \Delta (V(P) \setminus \{s, t\})$  of  $M_1$  and  $M_2$  is strictly larger than  $I$ . It is easy to see that  $D(I)$  has a directed path from  $s$  to  $t$  without shortcuts if and only if  $D(I)$  has a directed path from  $s$  to  $t$ . The following lemma summarizes the above discussion and also proves that the converse direction also holds.

► **Lemma 5** (Corollary 3.2 in [23]). *Let  $I$  be a common independent set in two matroids  $M_1$  and  $M_2$  and  $D(I)$  be the auxiliary directed graph. Then,  $I$  is a maximum common independent set in  $M_1$  and  $M_2$  if and only if  $D(I)$  has no directed  $s$ - $t$  path.*

Such a path  $P$  in  $D(I)$  is called an *augmenting path*. Lemma 5 is helpful to design an algorithm for enumerating all large maximal common independent sets in two matroids.

### 3 Enumeration of maximum common independent sets

We first consider the problem of enumerating all *maximum* common independent sets of two matroids, which is indeed a special case of LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION, where  $\tau = \text{opt}$ .<sup>2</sup> It is known that this problem can be solved in amortized polynomial time using the algorithm in [12]. However, an analysis of the delay of this algorithm is not explicitly given in their paper. In order to show an explicit delay bound, we give a polynomial-delay algorithm for MAXIMUM COMMON INDEPENDENT SET ENUMERATION, using a simple flashlight search technique (also known as binary partition and backtracking) [2, 27].

In this technique, an algorithm enumerates solutions by recursively picking one element  $e$  in  $S$  and partitioning the set of solutions into two subsets; One set consists of solutions including  $e$ , and the other set consists of solutions excluding  $e$ . After partitioning according to  $e$ , the algorithm repeats this partitioning process until all the elements in  $S$  are picked. It is easy to see that each solution set obtained in this process contains at most one solution. If a solution set contains a solution, then we just output it. To upper bound the running time of this recursive algorithm, we need to check whether a current solution set is empty

<sup>2</sup> By  $\text{opt}$ , we mean the maximum cardinality of a common independent set of  $M_1$  and  $M_2$ .

in the recursive partition process. We call such a subproblem an *extension problem*. To enumerate maximum common independent sets of matroids, we define MAXIMUM COMMON INDEPENDENT SET EXTENSION as follows.

► **Definition 6.** *Given two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$ , and two disjoint subsets  $In, Ex \subseteq S$ . MAXIMUM COMMON INDEPENDENT SET EXTENSION asks to find a maximum common independent set  $I$  of  $M_1$  and  $M_2$  that satisfies  $In \subseteq I$  and  $Ex \cap I = \emptyset$ .*

In what follows, we call conditions  $In \subseteq I$  and  $Ex \cap I = \emptyset$  the *inclusion condition* and *exclusion condition*, respectively. Note that for any matroid  $M$ ,  $M \setminus Ex$  and  $M / In$  are matroids. The following proposition is straightforward but essential for solving the extension problem.

► **Proposition 7.** *Let  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$  be two matroids, and  $In$  and  $Ex$  be disjoint subsets of  $S$ . Suppose that  $In$  is a common independent set of  $M_1$  and  $M_2$ . Let  $M'_1 = (M_1 / In) \setminus Ex$  and  $M'_2 = (M_2 / In) \setminus Ex$ . Then, there is a maximum common independent set  $I$  of  $M_1$  and  $M_2$  that satisfies  $In \subseteq I$  and  $Ex \cap I = \emptyset$  if and only if there is a common independent set  $I'$  of  $M'_1$  and  $M'_2$  with the cardinality  $|I| - |In|$ .*

By the above proposition, we can solve MAXIMUM COMMON INDEPENDENT SET EXTENSION in polynomial time by using a polynomial-time algorithm for finding a maximum common independent set of two matroids [28]. Note that by using oracles for  $M_1$  and  $M_2$ , we can check whether a subset of  $S$  is independent in  $M'_1$  and in  $M'_2$  in time  $O(n + Q)$  and space  $O(n + \hat{Q})$ .

Now, we design a simple flashlight search algorithm, which is sketched as follows. Let  $\mathcal{S}(In, Ex)$  be the set of maximum common independent sets of  $M_1$  and  $M_2$  that satisfy both the inclusion and exclusion conditions. Clearly, the set of all maximum common independent sets of  $M_1$  and  $M_2$  corresponds to  $\mathcal{S}(\emptyset, \emptyset)$ . By solving the extension problem, we can determine whether  $\mathcal{S}(In, Ex)$  is empty or not in polynomial time. Moreover, for an element  $e \in S \setminus (In \cup Ex)$ ,  $\{\mathcal{S}(In \cup \{e\}, Ex), \mathcal{S}(In, Ex \cup \{e\})\}$  is a partition of  $\mathcal{S}(In, Ex)$ . Thus, we can enumerate all maximum common independent sets in  $\mathcal{S}(In, Ex)$  by recursively enumerating  $\mathcal{S}(In \cup \{e\}, Ex)$  and  $\mathcal{S}(In, Ex \cup \{e\})$ . We give a pseudo-code of our algorithm in Algorithm 1. Finally, we consider the delay of this algorithm. Let  $\mathcal{T}$  be a recursion tree defined by the execution of Algorithm 1. As we output a maximum common independent set of  $M_1$  and  $M_2$  at each leaf node in  $\mathcal{T}$ , the delay of the algorithm is upper bounded by the “distance” of two leaf nodes times the running time required to processing each node in  $\mathcal{T}$ . The distance between the root and a leaf node of  $\mathcal{T}$  is at most  $n$  and thus, the distance between two leaf nodes in  $\mathcal{T}$  is upper bounded by linear in  $n$ . The time complexity of each node in  $\mathcal{T}$  is bounded by  $O(\text{poly}(n))$ . Hence, the delay of Algorithm 1 is polynomial in  $n$ . By using an  $O(\text{opt}^{3/2}nQ)$ -time and  $O(n^2 + \hat{Q})$ -space algorithm for finding a maximum common independent set of two matroids [7], the following theorem follows.

► **Theorem 8.** *We can enumerate all maximum common independent sets of  $M_1$  and  $M_2$  in  $O(\text{opt}^{3/2}n^2Q)$  delay and  $O(n^2 + \hat{Q})$  space.*

#### 4 Enumeration of large maximal common independent sets

We propose a polynomial-delay and polynomial-space algorithm for enumerating maximal common independent sets of two matroids  $M_1$  and  $M_2$  with the cardinality at least  $\tau$ , namely LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION. From Theorem 8, if  $\tau = \text{opt}$ , we can enumerate all maximum common independent sets of  $M_1$  and  $M_2$  in polynomial delay and polynomial space. Thus, in this section, we assume that  $\tau < \text{opt}$ .

■ **Algorithm 1** A polynomial-delay and polynomial-space algorithm for enumerating all maximum common independent sets in two matroids.

---

```

1 Procedure Maximum( $M_1 = (S, \mathcal{I}_1), M_2 = (S, \mathcal{I}_2)$ )
2   | RecMaximum( $M_1, M_2, \emptyset, \emptyset$ )
3 Procedure RecMaximum( $M_1, M_2, In, Ex$ )
4   | if  $In \cup Ex = S$  then Output  $In$ , return
5   | Choose an arbitrary  $e \in S \setminus (In \cup Ex)$ 
6   | if there is a maximum common independent set  $I'$  of  $M_1$  and  $M_2$  that satisfies
   |   both  $(In \cup \{e\}) \subseteq I'$  and  $Ex \cap I' = \emptyset$  then
7   |   | RecMaximum( $M_1, M_2, In \cup \{e\}, Ex$ )
8   | if there is a maximum common independent set  $I'$  of  $M_1$  and  $M_2$  that satisfies
   |   both  $In \subseteq I'$  and  $(Ex \cup \{e\}) \cap I' = \emptyset$  then
9   |   | RecMaximum( $M_1, M_2, In, Ex \cup \{e\}$ )

```

---

Our proposed algorithm is based on *reverse search* [1], which is one of the frequently used techniques to design efficient enumeration algorithms [14, 19, 21, 24, 29]. One may expect that a flashlight search algorithm similar to that described in the previous section could be designed for LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION, because finding a *maximal* solution is usually easier than finding a *maximum* solution. However, this intuitive phenomenon does not hold for extension problems. In particular, the problem of finding a maximal matching in a bipartite graph that satisfies an exclusion condition is NP-complete [5, 19]. As the set of all matchings in a bipartite graph can be described as the set of common independent sets of matroids, the extension problem for LARGE MAXIMAL COMMON INDEPENDENT SET ENUMERATION is NP-complete.

Before delving into our algorithm, we briefly sketch an overview of the reverse search technique. Let  $\mathcal{S}$  be the set of solutions. In the reverse search technique, we define a set of “special solutions”  $\mathcal{R} \subseteq \mathcal{S}$  and a rooted forest (i.e., a set of rooted trees) on  $\mathcal{S}$  whose roots belong to  $\mathcal{R}$ . Suppose that we can enumerate  $\mathcal{R}$  efficiently. Then, we can enumerate all solutions in  $\mathcal{S}$  by solely traversing the rooted forest from each root solution in  $\mathcal{R}$ . To this end, for a non-root solution  $X$  in  $\mathcal{S} \setminus \mathcal{R}$ , it suffices to define its parent  $\text{par}(X)$  in an appropriate manner. More specifically, to define a rooted forest on  $\mathcal{S}$ , this parent-child relation must have no cycles. Moreover, to traverse this rooted forest, we need to efficiently enumerate the children of each internal node in the rooted forest.

Now, we turn back to our problem. In the following, we may simply refer to maximal common independent sets of  $M_1$  and  $M_2$  with cardinality at least  $\tau$  as *solutions*. We define the set of maximum common independent sets of  $M_1$  and  $M_2$  as the root solutions  $\mathcal{R}$ . We can efficiently enumerate  $\mathcal{R}$  by Theorem 8. To define the parent of a solution not in  $\mathcal{R}$ , fix an arbitrary maximum common independent set  $R$  of  $M_1$  and  $M_2$ . Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$ . We consider two matroids  $M_1(R, I) := (M_1 \mid (R \cup I)) / (R \cap I)$  and  $M_2(R, I) := (M_2 \mid (R \cup I)) / (R \cap I)$  as well as the auxiliary directed graph  $D(R, I) := D_{M_1(R, I), M_2(R, I)}(I \setminus R)$ . Let us note that the vertex set of  $D(R, I)$  is  $(R \triangle I) \cup \{s, t\}$ . Since  $R$  and  $I$  are independent in both  $M_1$  and  $M_2$ , by Proposition 1,  $R \setminus I$  and  $I \setminus R$  are independent in  $M_1(R, I)$  and  $M_2(R, I)$  as well. Moreover, as  $|R| > |I|$ , we have  $|R \setminus I| > |I \setminus R|$ . Thus, by Lemma 5,  $D(R, I)$  has a directed  $s$ - $t$  path. Let  $P$  be a directed  $s$ - $t$  path  $(v_1 = s, v_2, \dots, v_{2k+1} = t)$  in  $D(R, I)$  without shortcuts. We first show that  $I \triangle \{v_2, v_3\}$  is a common independent set of  $M_1$  and  $M_2$ .

► **Lemma 9.** *Let  $P = (v_1 = s, \dots, v_{2k+1} = t)$  be a directed  $s$ - $t$  path without shortcuts in  $D(R, I)$ . Then  $P$  has at least four vertices and  $I \triangle \{v_2, v_3\}$  is also a common independent set of  $M_1$  and  $M_2$ .*

**Proof.** We first show that  $P$  has at least four vertices. As  $s$  is not adjacent to  $t$  in  $D(R, I)$ ,  $P$  has at least three vertices. If  $P = (v_1 = s, v_2, v_3 = t)$ , then  $(I \setminus R) \cup \{v_2\}$  is a common independent set of  $M_1(R, I)$  and  $M_2(R, I)$ . However, by Proposition 2,  $I \setminus R$  is a maximal common independent set of  $M_1(R, I)$  and  $M_2(R, I)$ , a contradiction.

We next show that  $I \triangle \{v_2, v_3\}$  is also a common independent set of  $M_1$  and  $M_2$ . By the definition of  $D(R, I)$ ,  $v_2 \in R \setminus I$  and  $v_3 \in I \setminus R$ . This implies that  $v_2 \notin I$  and  $v_3 \in I$ . Moreover, since  $D(R, I)$  has arcs  $(s, v_2)$  and  $(v_2, v_3)$ ,  $(I \setminus R) \cup \{v_2\}$  is independent in  $M_1(R, I)$  and  $(I \setminus R) \triangle \{v_2, v_3\}$  is independent in  $M_2(R, I)$ . By Proposition 1,  $I \cup \{v_2\}$  and  $I \triangle \{v_2, v_3\}$  are independent in  $M_1$  and in  $M_2$ , respectively. As  $I \triangle \{v_2, v_3\} = (I \setminus \{v_3\}) \cup \{v_2\}$  is a subset of  $I \cup \{v_2\}$ ,  $I \triangle \{v_2, v_3\}$  is a common independent set of  $M_1$  and  $M_2$ . ◀

We define the parent  $\text{par}(I)$  of  $I$  as follows. To ensure the consistency of defining its parent, we choose a path  $P$  from  $s$  to  $t$  without shortcuts in  $D(R, I)$  in a certain way, and hence the path  $P$  is determined solely by the pair  $R$  and  $I$ . We define the parent of  $I$  (under  $R$ ) as  $\mu(I \triangle \{v_2, v_3\})$ , where  $\mu(X)$  is an arbitrary maximal common independent set of  $M_1$  and  $M_2$  containing  $X$  for a common independent set  $X$  of  $M_1$  and  $M_2$ . Similarly, we choose a maximal common independent set  $\mu(X)$  in a certain way, and hence  $\mu(X)$  is determined solely by  $X$ . Therefore, by Lemmas 5 and 9, for a maximal common independent set  $I$  of  $M_1$  and  $M_2$  with  $|I| < |R|$ , the parent of  $I$  (under  $R$ ) is uniquely determined and we denote it as  $\text{par}(I)$ . In the following, we claim that this parent-child relation defines a rooted forest on the solutions whose roots belong to  $\mathcal{R}$ . A key to this is a certain “monotonicity”, which will be proven in Lemma 11: For any solution  $I$  with  $|I| < |R|$ , it holds that  $|R \triangle I| > |R \triangle \text{par}(I)|$ . Given these, from any solution  $I$  with  $|I| < |R|$ , we can “reach” a maximum common independent set of  $M_1$  and  $M_2$  (not necessarily to be  $R$ ) by iteratively taking its parent at most  $n$  times as  $|R \triangle I| \leq n$ . To show this monotonicity, we give the following technical lemma, whose proof is deferred to the end of this section.

► **Lemma 10.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$  and  $e \in I$  and  $f \in S \setminus I$ . If  $D(I)$  has two arcs  $(s, f)$  and  $(f, e)$ , then  $I \triangle \{e, f\}$  is a common independent set of  $M_1$  and  $M_2$ . Moreover,  $|\mu(I \triangle \{e, f\})| \leq |I| + 1$ .*

Now, we prove the aforementioned “monotonicity”.

► **Lemma 11.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$ . Then, the following three properties on  $\text{par}(I)$  are satisfied.*

- $|I| \leq |\text{par}(I)|$ ,
- $|R \triangle I| > |R \triangle \text{par}(I)|$ , and
- $|I \triangle \text{par}(I)| \leq 3$ .

**Proof.** As  $|R \setminus I| > |I \setminus R|$ , by Lemma 5, there is a directed path  $P = (v_1, \dots, v_{2k+1})$  from  $s = v_1$  to  $t = v_{2k+1}$  without shortcuts in  $D(R, I)$ . By Lemma 9,  $I' = I \triangle \{v_2, v_3\}$  is a common independent set of  $M_1$  and  $M_2$ . The first property  $|I| \leq |\text{par}(I)|$  follows from

$$|\text{par}(I)| = |\mu(I \triangle \{v_2, v_3\})| \geq |I \triangle \{v_2, v_3\}| = |I|,$$

as  $v_2 \notin I$  and  $v_3 \in I$ . Since  $v_2 \in R \setminus I$  and  $v_3 \in I \setminus R$ , we have  $|R \triangle I'| = |R \triangle I| - 2$ . If  $D(I)$  has arcs  $(s, v_2)$  and  $(v_2, v_3)$ , by Lemma 10,  $|\mu(I')| \leq |I| + 1$ , which yields that

$$|R \triangle \text{par}(I)| = |R \triangle \mu(I')| \leq |R \triangle I'| + 1 = |R \triangle I| - 1,$$

where the inequality  $|R \triangle \mu(I')| \leq |R \triangle I'| + 1$  follows from  $|\mu(I')| \leq |I| + 1 = |I'| + 1$ , meaning that  $\mu(I') \setminus I'$  contains at most one element. This also shows that

$$|I \triangle \text{par}(I)| = |I \triangle \mu(I')| \leq |I \triangle I'| + 1 \leq |I \triangle (I \triangle \{v_2, v_3\})| + 1 = 3.$$

Thus, it suffices to show that  $D(I)$  has these arcs  $(v_2, v_3)$  and  $(s, v_2)$ . Let  $M_1(R, I) = (M_1 \mid (R \cup I)) / (R \cap I)$  and  $M_2(R, I) = (M_2 \mid (R \cup I)) / (R \cap I)$ . As  $D(R, I)$  has the arc  $(v_2, v_3)$ ,  $(I \setminus R) \cup \{v_2\}$  and  $((I \setminus R) \cup \{v_2\}) \setminus \{v_3\}$  are dependent and independent in  $M_2(R, I)$ , respectively. By Proposition 1,

$$\begin{aligned} ((I \setminus R) \cup \{v_2\}) \cup (R \cap I) &= I \cup \{v_2\} \text{ and} \\ (((I \setminus R) \cup \{v_2\}) \setminus \{v_3\}) \cup (R \cap I) &= (I \cup \{v_2\}) \setminus \{v_3\} \end{aligned}$$

are dependent and independent in  $M_2$ , respectively. This implies that  $D(I)$  has an arc  $(v_2, v_3)$ . A similar argument for  $(s, v_2)$  and  $M_1(R, I)$  proves that  $D(I)$  has arc  $(s, v_2)$ , completing the proof of this lemma.  $\blacktriangleleft$

Now, we are ready to describe our algorithm, which is also shown in Algorithm 2. We assume that the size of a maximum common independent set of  $M_1$  and  $M_2$  is at least  $\tau$  as otherwise we do nothing. We first enumerate the set  $\mathcal{R}$  all maximum common independent sets of  $M_1$  and  $M_2$ . This can be done in polynomial delay and polynomial space using the algorithm in Theorem 8. We choose an arbitrary  $R \in \mathcal{R}$  and for each  $I \in \mathcal{R}$ , we enumerate all solutions that belong to the component containing  $I$  in the rooted forest defined by the parent-child relation. This is done by calling  $\text{RecMaximal}(M_1, M_2, I, R, \tau)$ . The procedure  $\text{RecMaximal}(M_1, M_2, I, R, \tau)$  recursively generates solutions  $I'$  with  $I = \text{par}(I')$ . We would like to emphasize that the algorithm only generates solutions  $I'$  with  $|I'| \geq \tau$ .

We first claim that all the solutions are generated by this algorithm. To see this, consider an arbitrary solution  $I$ . Define a value  $v(I)$  as

$$v(I) = \begin{cases} 0 & \text{if } |I| = |R| \\ |I \triangle R| & \text{otherwise.} \end{cases}$$

We prove the claim by induction on  $v(I)$ . Suppose that  $v(I) = 0$ . In this case,  $I$  is a maximum common independent set of  $M_1$  and  $M_2$ , as  $|I \triangle R| = 0$  if and only if  $I = R$ . Then,  $I$  is obviously generated as we call  $\text{RecMaximal}(M_1, M_2, I, R, \tau)$  for all  $I \in \mathcal{R}$ . Suppose that  $v(I) > 0$ . Then,  $I$  is a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$ . We assume that all the solutions  $I'$  with  $v(I') = |R \triangle I'| < |R \triangle I|$  is generated by the algorithm. By Lemma 11, we have  $|\text{par}(I)| \geq |I| \geq \tau$  and  $|R \triangle \text{par}(I)| < |R \triangle I|$ , which implies that  $\text{par}(I)$  is generated by the algorithm. By the definition of parent, we have  $\text{par}(I) = \mu(I \triangle \{u, v\})$  and  $|\text{par}(I) \triangle I| \geq 2$  for some  $u, v \in S$ . Moreover, by Lemma 11,  $|I \triangle \text{par}(I)| \leq 3$ , the child  $I$  of  $\text{par}(I)$  is computed at line 7. Thus,  $I$  is generated by the algorithm as well.

We next claim that all the solutions are generated without duplication. Since we only call  $\text{RecMaximal}(M_1, M_2, I', R, \tau)$  for  $I = \text{par}(I')$ , it holds that  $v(I) < v(I')$ . As  $v(I) \leq n$  for every solution  $I$ , by the uniqueness of the parent of non-maximum solutions, the algorithm generates each solution exactly once. This concludes that the algorithm correctly enumerates all maximal common independent sets of  $M_1$  and  $M_2$  of cardinality at least  $\tau$ .

Finally, we discuss the running time of the algorithm. We first enumerate all maximum common independent sets of  $M_1$  and  $M_2$ . This can be done in time  $O(n^{7/2}Q)$  delay. For each solution  $I$ , we compute  $\text{par}(I)$  as follows. We construct the graph  $D(R, I)$  that has  $n + 2$  nodes and  $O(n^2)$  arcs. This can be done by using  $O(n^2Q)$  queries to the oracles for

■ **Algorithm 2** A polynomial-delay and polynomial-space algorithm for enumerating all maximal common independent sets of  $M_1$  and  $M_2$  with the cardinality at least  $\tau$ .

---

```

1 Procedure Maximal( $M_1 = (S, \mathcal{I}_1), M_2 = (S, \mathcal{I}_2), \tau$ )
2   | Choose arbitrary  $R \in \mathcal{R}$ 
3   | foreach  $I \in \mathcal{R}$  do RecMaximal( $M_1, M_2, I, R, \tau$ )           // Use Algorithm 1.
4 Procedure RecMaximal( $M_1, M_2, I, R, \tau$ )
5   | Output  $I$ 
6   | foreach  $X \in \binom{S}{3} \cup \binom{S}{2}$  do
7   |   |  $I' \leftarrow I \Delta X$ 
8   |   | if  $I'$  is a maximal common independent set of  $M_1$  and  $M_2$  such that
9   |   |   |  $\tau \leq |I'| < |R|$  and  $I = \text{par}(I')$  then
9   |   |   |   | RecMaximal( $M_1, M_2, I', R, \tau$ )

```

---

$M_1$  and  $M_2$ . To find the path  $P$  from  $s$  to  $t$  without shortcuts, we just compute a shortest path from  $s$  to  $t$ , which can be done in  $O(n^2)$  time. Thus, we can compute  $I \Delta \{v_2, v_3\}$  in  $O(n^2Q)$  time as well. From  $I \Delta \{v_2, v_3\}$ ,  $\mu(I \Delta \{v_2, v_3\})$  can be computed in  $O(nQ)$  time. Thus, we can compute  $\text{par}(I)$  from  $I$  in time  $O(n^2Q)$ .

For each call  $\text{RecMaximal}(M_1, M_2, I, R, \tau)$ , we output exactly one solution. Moreover, the running time of computing all children of  $I$  is  $O(n^5Q)$ . This can be seen as there are  $O(n^3)$  candidates  $I'$  of children and we can check in  $O(n^2Q)$  time whether a candidate  $I'$  is in fact a child of  $I$ . Thus the delay of the algorithm is upper bounded by the time elapsed between two consecutive calls. As the depth of the rooted forest defined by recursive calls is at most  $n$ , this can be upper bounded by  $O(n^6Q)$ . As for the space complexity, by Theorem 8, we can enumerate all maximum common independent sets of  $M_1$  and  $M_2$  in  $O(n^2 + \hat{Q})$  space. In  $\text{RecMaximal}$ , we need to store local variables  $I$  and  $X$  in each recursive call, which can be done in space  $O(n)$ . As the depth of the rooted forest is at most  $n$ , the space usage for local variables is  $O(n^2)$  in total. For each candidate  $I'$ , we can check in  $O(n^2 + \hat{Q})$  whether  $I'$  is a maximal common independent set of  $M_1$  and  $M_2$  and whether  $I = \text{par}(I')$ . Overall, we have the following theorem.

► **Theorem 12.** *There is an  $O(n^6Q)$ -delay and  $O(n^2 + \hat{Q})$ -space algorithm for enumerating maximal common independent sets in two matroids with the cardinality at least  $\tau$ .*

#### 4.1 Proof of Lemma 10

To complete our proof of Theorem 12, we need to show the correctness of Lemma 10. To this end, we focus on  $D(I)$ . Since  $I$  is a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$ ,  $D(I)$  has a directed  $s$ - $t$  path  $P = (v_1 = s, v_2 = f, v_3 = e, \dots, v_{2k+1} = t)$ . By the definition of  $D(I)$ ,  $I \Delta \{e, f\}$  is a common independent set of  $M_1$  and  $M_2$ . We first show that  $I \Delta \{e, f\}$  becomes dependent in  $M_1$  when we add an element  $f'$  in  $N_{D(I)}^+(e)$ .

► **Lemma 13.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$ ,  $e$  be an element in  $I$ , and  $f_1$  and  $f_2$  be distinct two elements in  $N_{D(I)}^+(e)$ . Then,  $I' := (I \setminus \{e\}) \cup \{f_1, f_2\}$  is dependent in  $M_1$ .*

**Proof.** Since  $D(I)$  has arcs  $(e, f_1)$  and  $(e, f_2)$ , both  $(I \setminus \{e\}) \cup \{f_1\}$  and  $(I \setminus \{e\}) \cup \{f_2\}$  are independent in  $M_1$ , and  $I \cup \{f_1\}$  and  $I \cup \{f_2\}$  are dependent in  $M_1$ . Thus,  $M_1$  has two circuits  $C_1$  and  $C_2$  that contain  $\{e, f_1\}$  and  $\{e, f_2\}$ , respectively. By the circuit elimination axiom, there is a circuit  $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$ . Since  $I'$  contains  $(C_1 \cup C_2) \setminus \{e\}$ , it also contains  $C_3$  and hence is dependent in  $M_1$ . ◀

► **Lemma 14.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$ ,  $e$  be an element in  $I$ , and  $f_1$  and  $f_2$  be distinct two elements in  $N_{D(I)}^-(e)$ . Then,  $I' := (I \setminus \{e\}) \cup \{f_1, f_2\}$  is dependent in  $M_2$*

**Proof.** Since  $D(I)$  has arcs  $(f_1, e)$  and  $(f_2, e)$ , both  $I \Delta \{e, f_1\}$  and  $I \Delta \{e, f_2\}$  are independent in  $M_2$  and  $I \cup \{f_1\}$  and  $I \cup \{f_2\}$  are dependent in  $M_2$ . Thus,  $M_2$  has two circuits  $C_1$  and  $C_2$  that contain  $\{e, f_1\}$  and  $\{e, f_2\}$ , respectively. By the circuit elimination axiom, there is a circuit  $C_3 \subseteq (C_1 \cup C_2) \setminus \{e\}$ . Since  $I'$  contains  $(C_1 \cup C_2) \setminus \{e\}$ , it also contains  $C_3$  and hence is dependent in  $M_2$ . ◀

We show that  $\mu(I \Delta \{e, f\})$  does not contain any element in  $S \setminus (I \cup N_{D(I)}^+(e) \cup N_{D(I)}^-(e))$ .

► **Lemma 15.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$ ,  $e$  be an element in  $I$ , and  $f$  be an element in  $S \setminus (I \cup N_{D(I)}^+(e) \cup N_{D(I)}^-(e))$ . Then,  $I \Delta \{e, f\}$  is dependent in at least one of  $M_1$  or in  $M_2$ .*

**Proof.** From the maximality of  $I$ ,  $I \cup \{f\}$  is dependent in at least one of  $M_1$  or  $M_2$ . Suppose that  $I \cup \{f\}$  is dependent on  $M_2$ . Then,  $I \cup \{f\}$  contains at least one circuit  $C$  of  $M_2$  containing  $f$ . We show that  $I \cup \{f\}$  contains only one circuit of  $M_2$ . If  $I \cup \{f\}$  contains another circuit  $C'$  with  $f \in C'$ , by the circuit elimination axiom,  $(C \cup C') \setminus \{f\}$  contains a circuit, which contradicts the fact that  $I$  is independent in  $M_2$ . Thus,  $M_2$  has the unique circuit  $C$ , which is contained in  $I \cup \{f\}$ . Observe that  $N_{D(I)}^+(f) \cap I = C \setminus \{f\}$ , since  $(I \cup \{f\}) \setminus \{e\}$  is independent in  $M_2$  for  $e' \in C$  due to the minimality of  $C$ . As  $f \in S \setminus (I \cup N_{D(I)}^+(e) \cup N_{D(I)}^-(e))$ , we have  $e \notin C$ . Hence,  $(I \setminus \{e\}) \cup \{f\}$  contains  $C$ , that is,  $(I \setminus \{e\}) \cup \{f\}$  is dependent in  $M_2$ . When  $I \cup \{f\}$  is dependent in  $M_1$ ,  $(I \setminus \{e\}) \cup \{f\}$  is also dependent in  $M_1$  from a similar discussion. ◀

Now we are ready to prove Lemma 10.

► **Lemma 10.** *Let  $I$  be a maximal common independent set of  $M_1$  and  $M_2$  with  $|I| < |R|$  and  $e \in I$  and  $f \in S \setminus I$ . If  $D(I)$  has two arcs  $(s, f)$  and  $(f, e)$ , then  $I \Delta \{e, f\}$  is a common independent set of  $M_1$  and  $M_2$ . Moreover,  $|\mu(I \Delta \{e, f\})| \leq |I| + 1$ .*

**Proof.** By the definition of  $D(I)$ ,  $I \Delta \{e, f\}$  is a common independent set since  $D(I)$  has two arcs  $(s, f)$  and  $(f, e)$ . Thus,  $\mu(I \Delta \{e, f\})$  is a maximal common independent set of  $M_1$  and  $M_2$ . We show that  $|\mu(I \Delta \{e, f\})| \leq |I| + 1$ . Since  $f$  is contained in  $N_{D(I)}^-(e)$ ,  $\mu(I \Delta \{e, f\})$  does not contain elements in  $N_{D(I)}^-(e)$  except for  $f$  by Lemma 14. Moreover, by Lemma 13,  $\mu(I \Delta \{e, f\})$  contains at most one element in  $N_{D(I)}^+(e)$ . Finally,  $\mu(I \Delta \{e, f\})$  does not contain any element in  $S \setminus (I \cup N_{D(I)}^+(e) \cup N_{D(I)}^-(e))$  by Lemma 15. Therefore,  $\mu(I \Delta \{e, f\}) \setminus (I \Delta \{e, f\})$  contains at most one element. Since  $|I \Delta \{e, f\}| = |I|$ ,  $|\mu(I \Delta \{e, f\})| \leq |I| + 1$ . ◀

## 5 Ranked enumeration

In this section, we give a ranked enumeration algorithm for enumerating maximal common independent sets of two matroids  $M_1 = (S, \mathcal{I}_1)$  and  $M_2 = (S, \mathcal{I}_2)$ . Recall that an enumeration algorithm is called a *ranked enumeration algorithm* if the algorithm enumerates solutions in a non-increasing order of their cardinality. We do this in a slightly general manner.

In what follows, we consider the following abstract problem. Let  $S$  be a finite set and  $\mathcal{F}$  be a subset of  $2^S$ . Let  $\mathcal{A}(\tau)$  be an algorithm that outputs all sets in  $\mathcal{F}$  with the cardinality at least  $\tau$ . We denote the maximum delay complexity and the space complexity from  $\mathcal{A}(\tau)$  to  $\mathcal{A}(1)$  as  $t(n)$  and  $s(n)$ , respectively. Moreover, we denote the number of outputs of  $\mathcal{A}(\tau)$  as  $\#\mathcal{A}(\tau)$ . Under this problem setting, we construct a ranked enumeration algorithm that outputs the  $i$ -th solution in  $O(i \cdot n \cdot t(n))$  time with  $O(s(n))$  space as follows.

Our idea is simply to execute  $\mathcal{A}$  from  $\mathcal{A}(n)$  to  $\mathcal{A}(1)$ . When  $\mathcal{A}(k)$  outputs a solution with cardinality more than  $k$ , we just ignore it. In other words, when we execute  $\mathcal{A}(i)$ , all solutions in  $\mathcal{F}$  with cardinality exactly  $i$  are output. Clearly, we can enumerate all solutions in  $\mathcal{F}$  in a non-increasing order of their cardinality. We consider the time and space complexity of this method. It is easy to see that the space complexity of this algorithm is  $O(s(n))$  as we just execute  $\mathcal{A}$  in order. Thus, we estimate the running time required to output the first  $i$  solutions for  $i \leq |\mathcal{F}|$ . Let  $j \geq 1$  be the maximum integer such that  $\#\mathcal{A}(j)$  is less than  $i$ . Since the delay of  $\mathcal{A}$  is bounded by  $t(n)$  and  $\#\mathcal{A}(j-1)$  is at least  $i$ ,  $\mathcal{A}(j-1)$  outputs the  $i$ -th solution in  $O(i \cdot t(n))$  time. Since the total running time is bounded by  $O((n-j+1) \cdot \#\mathcal{A}(j) \cdot t(n) + i \cdot t(n)) = O(i \cdot n \cdot t(n))$  time, this algorithm outputs the first  $i$  solutions in  $O(i \cdot n \cdot t(n))$  time.

► **Theorem 16.** *Let  $S$  be a finite set and  $\mathcal{F}$  be a subset of  $2^S$ . For any  $1 \leq k \leq \tau$ , suppose that we have an algorithm  $\mathcal{A}(k)$  that enumerates all sets in  $\mathcal{F}$  with the cardinality at least  $k$  for any  $1 \leq k \leq \tau$  in  $t(n)$  delay and  $s(n)$  space. Then, there is an algorithm enumerating all subsets in  $\mathcal{I}$  in non-increasing order of their cardinality that outputs the first  $i$  solutions in  $O(i \cdot n \cdot t(n))$  time using  $O(s(n))$  space for  $i \leq |\mathcal{F}|$ .*

We obtain a linear incremental-time and polynomial-space ranked enumeration algorithm for maximal common independent sets of two matroids by combining Theorems 12 and 16.

► **Theorem 17.** *There is a linear incremental-time and polynomial-space algorithm for enumerating all maximal common independent sets in two matroids in non-increasing order. This algorithm outputs the first  $i$  solutions in  $O(i \cdot n^7 Q)$  time.*

## 6 Applications of our algorithms

Due to an expressive power of MATROID INTERSECTION, Theorems 12 and 17 give enumeration algorithms for various combinatorial objects in a unified way. An example of such objects is to enumerate maximal  $b$ -matchings in bipartite graphs. It is known that an intersection of two matroids can represent all objects in the following theorem. See the appendix for details on representing these objects by an intersection of two matroids.

► **Theorem 18.** *There are polynomial delay and space enumeration algorithms for*

- *maximal bipartite  $b$ -matchings with cardinality at least  $\tau$ ,*
- *maximal colorful forests with cardinality at least  $\tau$ , and*
- *maximal degree constrained subgraphs in digraphs with cardinality at least  $\tau$ ,*

*Moreover, there are linear incremental-time and polynomial-space ranked enumeration algorithms for the above problems.*

---

### References

- 1 D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996. doi:10.1016/0166-218X(95)00026-N.
- 2 Etienne Birmelé, Rui Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1884–1896. SIAM, 2013. doi:10.1137/1.9781611973105.134.
- 3 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Matroid intersections, polymatroid inequalities, and related problems. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002*, pages 143–154, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

- 4 Florent Capelli and Yann Strozecki. Geometric Amortization of Enumeration Algorithms. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2023.18.
- 5 Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Extension of some edge graph problems: Standard and parameterized complexity. In *Proc. FCT 2019*, volume 11651 of *Lecture Notes in Computer Science*, pages 185–200. Springer, 2019. doi:10.1007/978-3-030-25027-0\_13.
- 6 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Listing maximal independent sets with minimal space and bounded delay. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2017. doi:10.1007/978-3-319-67428-5\_13.
- 7 William H. Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986. doi:10.1137/0215066.
- 8 Shaleen Deep and Paraschos Koutris. Ranked Enumeration of Conjunctive Query Results. In Ke Yi and Zhewei Wei, editors, *24th International Conference on Database Theory (ICDT 2021)*, volume 186 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICDT.2021.5.
- 9 Jack Edmonds. Paths, trees, and flowers. *Can. J. Math.*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.
- 10 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications, (Proceedings, Calgary International Conference on Combinatorial Structures and Their Applications)*, pages 69–87, Calgary, Alberta, 1970.
- 11 David Eppstein.  $k$ -Best Enumeration. In *Encyclopedia of Algorithms*, pages 1003–1006. Springer, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4\_733.
- 12 Komei Fukuda and Makoto Namiki. Finding all common bases in two matroids. *Discrete Applied Mathematics*, 56(2):231–243, 1995. Fifth Franco-Japanese Days. doi:10.1016/0166-218X(94)00088-U.
- 13 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Inform. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.
- 14 Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial Delay Algorithm for Listing Minimal Edge Dominating Sets in Graphs. In *Proc. of WADS 2015*, pages 446–457, 2015. doi:10.1007/978-3-319-21840-3\_37.
- 15 L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, and K. Makino. Enumerating spanning and connected subsets in graphs and matroids. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006*, pages 444–455, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 16 Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating Cut Junctions in Graphs and Related Problems. *Algorithmica*, 51(3):239–263, 2008. doi:10.1007/s00453-007-9111-9.
- 17 Leonid G. Khachiyan, Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM J. Discret. Math.*, 19(4):966–984, 2005. doi:10.1137/S0895480103428338.

- 18 Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa. Efficient constant-factor approximate enumeration of minimal subsets for monotone properties with weight constraints. *CoRR*, abs/2009.08830, 2020. [arXiv:2009.08830](#).
- 19 Yasuaki Kobayashi, Kazuhiro Kurita, and Kunihiro Wasa. Polynomial-delay and polynomial-space enumeration of large maximal matchings. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science*, pages 342–355, Cham, 2022. Springer International Publishing.
- 20 Tuukka Korhonen. Listing small minimal separators of a graph. *CoRR*, abs/2012.09153, 2020. [arXiv:2012.09153](#).
- 21 Kazuhiro Kurita and Yasuaki Kobayashi. Efficient enumerations for minimal multicuts and multiway cuts. In *Proc. MFCS 2020*, pages 60:1–60:14, 2020. [doi:10.4230/LIPIcs.MFCS.2020.60](#).
- 22 E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating All Maximal Independent Sets: NP-Hardness and Polynomial-Time Algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.
- 23 Eugene L Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9(1):31–56, 1975. [doi:10.1007/BF01681329](#).
- 24 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Proc. SWAT 2004*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2004. [doi:10.1007/978-3-540-27810-8\\_23](#).
- 25 James Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.
- 26 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked Enumeration of Minimal Triangulations. In *Proc. of PODS 2019*, pages 74–88, 2019. [doi:10.1145/3294052.3319678](#).
- 27 R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975. [doi:10.1002/net.1975.5.3.237](#).
- 28 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.
- 29 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM J. Comput.*, 6(3):505–517, 1977. [doi:10.1137/0206036](#).
- 30 Takeaki Uno. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In Hon Wai Leong, Hiroshi Imai, and Sanjay Jain, editors, *Algorithms and Computation*, pages 92–101, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
- 31 Takeaki Uno. A fast algorithm for enumerating bipartite perfect matchings. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation*, pages 367–379, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

# Formalizing Hyperspaces for Extracting Efficient Exact Real Computation

Michal Konečný 

School of Computing, Aston University, UK

Sewon Park 

Research Institute for Mathematical Sciences, Kyoto University, Japan

Holger Thies 

Graduate School of Human and Environmental Science, Kyoto University, Japan

---

## Abstract

We propose a framework for certified computation on hyperspaces by formalizing various higher-order data types and operations in a constructive dependent type theory. Our approach builds on our previous work on axiomatization of exact real computation where we formalize nondeterministic first-order partial computations over real and complex numbers. Based on the axiomatization, we first define open, closed, compact and overt subsets in an abstract topological way that allows short and elegant proofs with computational content coinciding with standard definitions in computable analysis. From these proofs we extract programs for testing inclusion, overlapping of sets, et cetera.

To improve extracted programs, our framework specializes the Euclidean space  $\mathbb{R}^m$  making use of metric properties. To define interesting operations over hyperspaces of Euclidean space, we introduce a nondeterministic version of a continuity principle valid under the standard type-2 realizability interpretation. Instead of choosing one of the usual formulations, we define it in a way similar to an interval extension operator, which often is already available in exact real computation software.

We prove that the operations on subsets preserve the encoding, and thereby define a small calculus to build new subsets from given ones, including limits of converging sequences with regards to the Hausdorff metric. From the proofs, we extract programs that generate drawings of subsets of  $\mathbb{R}^m$  with any given precision efficiently. As an application we provide a function that constructs fractals, such as the Sierpinski triangle, from iterated function systems using the limit operation, resulting in certified programs that errorlessly draw such fractals up to any desired resolution.

**2012 ACM Subject Classification** Theory of computation → Logic and verification

**Keywords and phrases** Computable analysis, type theory, program extraction

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.59

**Supplementary Material** *Software*: <https://github.com/holgerthies/coq-aern>  
archived at `swh:1:dir:6bfc91ee518681f328e69d08125f48ac0db1ffad`

**Funding** This project has received funding from the EU's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

*Sewon Park*: Supported by JSPS KAKENHI (Grant-in-Aid for JSPS Fellows) JP22F22071 as a JSPS International Research Fellow.

*Holger Thies*: Supported by JSPS KAKENHI Grant Numbers JP20K19744 and JP23H03346.

**Acknowledgements** The authors thank Pieter Collins, Arno Pauly and Hideki Tsuiki for interesting discussions.

## 1 Introduction

In exact real computation, real numbers are often presented as abstract data type hiding tedious multi-precision computations used to eliminate rounding errors in the background from the user. Extensionally to the user, the real numbers resemble the classical structure of real



© Michal Konečný, Sewon Park, and Holger Thies;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 59; pp. 59:1–59:16

Leibniz International Proceedings in Informatics



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

numbers which makes reasoning procedures more intuitive, emancipating the programmers from tracking artificial numerical errors. The theoretical framework to study computability of algorithms in this context is computable analysis [16, 28]. Recent developments in implementations demonstrate the usefulness of this approach in practice [1, 20, 15].

The project cAERN [14] aims to provide an axiomatic formalization of exact real number computations in a constructive type theory, modeling real numbers similar as observed by users of exact real computation software; cf., [2, 25] where explicit representations are required for computations. Instead of being constructed as sequences of approximations, real numbers are axiomatized in a way that two reals are provably equal when they represent the same numbers; cf., [3, 9, 26]. This can be regarded as a modification of the classical real numbers by replacing uncomputable operations (e.g., rounding to nearest integers) and computably invalid axioms (e.g., the law of trichotomy) with their computable variants. As the axiomatization is representation irrelevant, reals appear similar to the classical real numbers allowing many classical results to be transported into our setting (see [14, Section 6]). At the same time, by allowing only computably valid axioms, it admits a program extraction mechanism built on top of an existing exact real computation framework.

The extraction is achieved by mapping the axiomatic real type  $\mathbb{R}$  to the abstract data type of real numbers  $\mathbf{CReal}$  and axioms to primitive operations in AERN, a Haskell library for exact real number computation which is being developed by one of the authors [15]. Thus, we do not need to focus on the concrete representation of real numbers and its efficient implementation which is challenging on its own (see e.g. [19]).

More concretely, in our system a proof of a theorem of the form

$$\Pi(x : \mathbb{R}). M\Sigma(y : \mathbb{R}). P x y$$

yields a nondeterministic AERN function of type  $\mathbf{CReal} \rightarrow \mathbf{CReal}$ . Here,  $M : \mathbf{Type} \rightarrow \mathbf{Type}$  formalizes nondeterminism, which is an inevitable effect of identifying real numbers extensionally [17].<sup>1</sup>

The theory formalized in cAERN is sufficient to build interesting first-order nondeterministic functions [13]. Many applications in exact real computation, e.g. reachability questions for dynamical systems, computing integrals and solving differential equations, however deal with higher order objects such as hyperspaces of functions and subsets as primitive data type [8]. Such applications are often safety-critical and a framework for formally proving correctness of algorithms on such higher-order objects is thus highly desirable.

In this paper, we extend our previous axiomatic formalization towards higher-order exact real computations. Based on the axiomatization of real numbers, partial lazy Boolean, and nondeterminism, we define various higher-order types including Euclidean spaces, classical subsets, open subsets, closed subsets, compact subsets and overt subsets. Furthermore, we provide various operations on these sets, including limit operations on compact sets based on Hausdorff distance. We further provide a rich theory of subsets of Euclidean spaces and operations on them, encoding them in a way that lets us efficiently generate verified drawings.

An important fact in computable analysis is that every computable function is continuous. When we operate on higher-order objects, this fact becomes computationally relevant. To make continuity available in our system, we include an axiomatic formalization of the continuity principle saying that any function from real numbers *nondeterministically* admits a modulus of continuity function. As the goal of the project is to use axioms to model

---

<sup>1</sup> Nondeterminism in this context is also called multivaluedness [28] or non-extensionality [5].

functionality that is typically available in exact real computation, instead of assuming a continuity principle by saying that any functional  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$  is continuous and then derive various continuity lemmas on  $X \rightarrow Y$  by reasoning on specific constructions of the types  $X$  and  $Y$ , decomposing them to the natural numbers, we formalize the continuity principle directly on our axiomatic types in terms of an interval extension. Such an operation is natural in exact real computation software and our continuity principle can be extracted to a simple operation in AERN.

Using the continuity principle, we show that our efficient encoding of euclidean subsets is computationally equivalent to a more general way to represent subsets. We further show that it can be used to generate verified drawings of images of functions on certain subsets.

As an application we devise a general construction of fractals defined by a recurrence relation on compact sets. The recurrence relation can be used to define the structure, which then can be drawn exactly up to any desired resolution using the drawing operation. We further can use the limit operator for a simple and elegant formulation of the theorem.

This paper is organized as follows. In Section 2 we briefly summarize the background from our previous work, introduce notations used throughout the paper and explain some extensions to the theory necessary for the current work. In Section 3 we define the continuity principle and prove some of its consequences. We define open, closed, compact and overt subsets in a generic setting and prove some of the properties in Section 4. We consider subsets of Euclidean space for which we use specialized encodings in Section 5 and finally show some applications to generate certified drawings of two-dimensional sets in Section 6.

All statements in this paper except for those in Appendix A have been implemented and fully formally verified in the Coq proof assistant as part of the cAERN library<sup>2</sup>. We extend the program extraction mechanism so that the newly added axioms, including the continuity principle, are mapped to appropriate AERN functions. The constructive content of the proofs can thus be extracted to AERN programs using Coq’s Haskell program extraction functionality and mapping axiomatically defined operations to basic operations in AERN providing a certified module in AERN for defining and manipulating higher-order objects.

## 2 Preliminaries

### 2.1 Type Theory and Realizability

As in [14] we assume to work in a simple type theory with basic types  $0, 1, 2, \mathbb{N}, \mathbb{Z}$ , a universe of classical propositions  $\mathbf{Prop}$ , and a universe of types  $\mathbf{Type}$ . We assume that the identity types  $=$  are in  $\mathbf{Prop}$  and that  $\mathbf{Prop}$  is a type universe closed under  $\rightarrow, \times, \vee, \exists, \Pi$ , containing two types  $\mathbf{True}, \mathbf{False} : \mathbf{Prop}$  which are the unit and the empty type respectively. We denote by  $P \vee Q : \mathbf{Prop}$  the classical fact that  $P$  or  $Q$  holds, and by  $P + Q : \mathbf{Type}$  the fact that there is a computational procedure deciding if  $P$  or  $Q$  holds. Similarly, when we have a family of classical propositions  $P : X \rightarrow \mathbf{Prop}$ , the type  $\exists(x : X). P x : \mathbf{Prop}$  denotes the classical existence of  $x : X$  satisfying  $P x$  while the ordinary dependent pair type (also called  $\Sigma$ -type),  $\Sigma(x : X). P x : \mathbf{Type}$  denotes the constructive existence.

We assume enough axioms that make  $\mathbf{Prop}$  indeed a universe of classical propositions and that are valid under this interpretation. That includes the law of excluded middle

$$\Pi(P : \mathbf{Prop}). P \vee \neg P$$

<sup>2</sup> <https://github.com/holgerthies/coq-aern>

where  $\neg P$  is defined by  $P \rightarrow \text{False}$  and propositional extensionality

$$\Pi(P, Q : \text{Prop}). (P \rightarrow Q) \rightarrow (Q \rightarrow P) \rightarrow P = Q.$$

These axioms allow us to reason classically when we are dealing with classical statements such as  $\exists(x : X). P x : \text{Prop}$ . Furthermore, we assume a general (dependent) function extensionality

$$\Pi(P : X \rightarrow \text{Type}). \Pi(f, g : \Pi(x : X). P x). (\Pi(x : X). f x = g x) \rightarrow f = g.$$

Based on this foundational setting of a constructive type theory we axiomatically formalize various types and operations for real number computation that are briefly introduced in the next subsections. The soundness of the basic setting of type theory and the newly introduced axioms are justified by extending a realizability interpretation in the category of assemblies over Kleene's second algebra [11, 23] by mapping each axiom to a computable operation in computable analysis.

## 2.2 Kleenean and Sierpinski Spaces

One important characteristic of exact real computation, besides exact operations, is nontermination. As real numbers are expressed exactly using infinite representations, comparing real numbers is a partial operation where testing  $x < y$  does not terminate when  $x = y$  regardless of the specific representations that are used [28, Theorem 4.1.16]. To deal with such partiality, instead of making computations continue indefinitely when the same numbers are compared, AERN and other exact real computation software provide a data type for lazy Boolean  $\{\text{ff}, \text{tt}, \perp\}$ , generalizing Booleans by adding a third element  $\perp$  as an explicit state of divergence. The third value  $\perp$  is an admissible value that a variable can store whose nontermination is delayed until it is required to test whether the value is  $\text{tt}$  or  $\text{ff}$ .

Observing that the space satisfies the algebraic structure of Kleene's three-valued logic where  $\perp$  is the third truth value for indeterminacy, the data type is also often called *Kleenean*. In particular, we assume that there are three distinct constants  $\text{true}, \text{false}, \text{bot} : \mathbb{K}$ , although we do not assume that they are the only elements by adding an induction rule. We write  $[k]$  for  $k = \text{true}$ .

Later we often need to deal with infinite sequences of Kleenean expressions. This has not yet been considered in our previous work and requires to characterize the Kleeneans further by assuming some additional axioms. The *nondeterministic countable choice principle* is an axiom of the following type:

$$\Pi(x : \mathbb{N} \rightarrow \mathbb{K}). (\exists(n : \mathbb{N}). [x\ n]) \rightarrow \text{M}\Sigma(n : \mathbb{N}). [x\ n]. \quad (1)$$

Here,  $\text{M}$  is a nondeterminism monad such that for any type  $X : \text{Type}$ , we automatically get a type  $\text{M}X : \text{Type}$  modelling the result of a nondeterministic computation in  $X$ . The axiom says whenever we have a sequence of Kleenean terms  $x$  and know that there (classically) is at least one index  $n$  such that  $x\ n$  is  $\text{true}$ , then we can nondeterministically find such an index.

Sometimes we only need to know if there is an element of the sequence that is  $\text{true}$ . For this, we add the following axiom.

$$\Pi(x : \mathbb{N} \rightarrow \mathbb{K}). \Sigma(k : \mathbb{K}). ([k] \leftrightarrow \exists(n : \mathbb{N}). [x\ n]).$$

That is, we can find a  $k : \mathbb{K}$  that is  $\text{true}$  iff any of the sequence elements is  $\text{true}$  and can be  $\text{false}$  or  $\text{bot}$  otherwise. To see that the axiom is valid, it suffices to show that such a  $k$  is computable which can be done by iterating over all the outputs of the realizers of  $x\ n$  in parallel.

A space similar to the Kleenean that is more commonly used in computability theory is the Sierpinski space. Sierpinski space is the topological space  $\{\downarrow, \uparrow\}$  whose only nontrivial open is  $\{\downarrow\}$ . In a certain way,  $\mathbf{K}$  can be seen as an extension of Sierpinski space by adding another decidable element `false`. However, as the type of Kleenean  $\mathbf{K}$  already exists in our system, we instead consider Sierpinski as a subtype of  $\mathbf{K}$  by defining it as

$$\mathbf{S} := \Sigma(b : \mathbf{K}). b \neq \text{false}$$

and name  $\downarrow := (\text{true}, t_{\downarrow})$  and  $\uparrow := (\text{bot}, t_{\uparrow})$  where the  $t_{\downarrow}$  is the unique proof term for `true`  $\neq$  `false` and  $t_{\uparrow}$  is the unique proof term for `⊥`  $\neq$  `false`. The uniqueness of the proof terms is due to the `Prop` axioms. From the computational point of view,  $\mathbf{S}$  can be thought of as  $\mathbf{K}$  where it is promised that `false` does not appear.

As a further axiom, we assume that the first projection  $\text{proj}_1 : \mathbf{S} \rightarrow \mathbf{K}$  admits a retraction  $\text{KtoS} : \mathbf{K} \rightarrow \mathbf{S}$ . Observe that the assumption can be validated by a program on the Kleenean that on its input  $b : \mathbf{K}$  checks if  $b = \text{true}$  or `false` and simply diverges when  $b = \text{false}$ .

### 2.3 Real Numbers and Euclidean Space

We assume real numbers by declaring that there is a type  $\mathbf{R}$  for real numbers containing two distinct constants 0 and 1, the standard arithmetical operators and a semi-decidable  $\mathbf{K}$ -valued comparison operator. From this axiomatization we can define further operations such as  $\max(x, y)$ ,  $|x|$ , etc. Again, the full axiomatization can be found in [14]. For any  $m : \mathbf{N}$  we further define the type  $\mathbf{R}^m$  simply as an  $m$ -element list of  $\mathbf{R}$  with maximum norm  $\|\cdot\|$ , and define the vector space operations point-wise.

We often approximate real numbers by elements of a countable dense subset. We call a type  $X$  enumerable if there exists a map  $f : \mathbf{N} \rightarrow X$  that is surjective in the classical sense. We define dyadic numbers by pairs  $\mathbf{D} := \mathbf{Z} \times \mathbf{N}$  and identify the pair  $(z, n)$  with the real  $z \cdot 2^{-n}$ . It is not hard to show that  $\mathbf{D}$  is enumerable. Similarly we write  $\mathbf{D}^m$  for the  $m$ -element list of  $\mathbf{D}$  when  $m : \mathbf{N}$  and suppose that the trivial coercion from  $\mathbf{D}^m$  to  $\mathbf{R}^m$  is taken implicitly.

We use dyadic numbers to approximate real numbers. In our theory, we can prove that  $\mathbf{D}$  is indeed dense in  $\mathbf{R}$  in the sense that for any real number  $x$ , we can nondeterministically find a dyadic rational approximation up to any prescribed error  $2^{-p}$ . More generally, we can prove the following statement in the Euclidean space:

$$\Pi(x : \mathbf{R}^m). \Pi(p : \mathbf{N}). \text{M}\Sigma(d : \mathbf{D}^m). \|x - d\| \leq 2^{-p}.$$

## 3 Continuity

Exact real computation software often internally represents real numbers by converging sequences of intervals with dyadic rational endpoints. Thus, for any function  $f : \mathbf{R} \rightarrow \mathbf{S}$ , whenever  $(f\ x)\downarrow$ , there is some dyadic interval containing  $x$  that made the software computing  $f\ x$  return  $\top$ . We thus formulate continuity on an abstract level similar to the above without referring to specific constructions of continuous types.

The Sierpinski space has two open subsets  $\{\downarrow\}$  and  $\{\downarrow, \uparrow\}$ . An interval extension of a function  $f : \mathbf{R} \rightarrow \mathbf{S}$  is a discrete function that indicates in which of the two open sets all values for  $f$  lie for any dyadic interval. As there are exactly two values, we can represent  $\{\downarrow\}$  and  $\{\downarrow, \uparrow\}$  space by the Boolean constants `true` and `false`, respectively. We can then see a map  $\hat{f}$  of type  $\mathbf{D} \times \mathbf{N} \rightarrow \mathbf{B}$  as a set-valued function from dyadic intervals:  $\hat{f}$  returns a subset of Sierpinski space on its argument  $(d, n) : \mathbf{D} \times \mathbf{N}$  representing the interval  $(d - 2^{-n}, d + 2^{-n}) \subset \mathbb{R}$ .

## 59:6 Formalizing Hyperspaces for Extracting Efficient Exact Real Computation

For a map  $f : \mathbb{R} \rightarrow \mathbb{S}$ , we say that a set-valued function on dyadic intervals  $\hat{f}$  is an interval extension of  $f$  when it satisfies the inclusion property:

$$\text{incl } \hat{f} := \Pi(d : \mathbb{D}). \Pi(n : \mathbb{N}). \hat{f}(d, n) = \text{true} \rightarrow (\Pi(x : \mathbb{R}). |x - d| \leq 2^{-n} \rightarrow (f x) \downarrow).$$

In words, whenever the interval extension  $\hat{f}$  of  $f$  returns true on the pair  $(d, n)$ ,  $(f x) \downarrow$  for all  $x \in (d - 2^{-n}, d + 2^{-n})$ .

However, not all interval extensions are interesting as the inclusion property holds also for the trivial map  $\hat{f}$  that returns false for all inputs. We say that a map  $\hat{f} : \mathbb{D} \times \mathbb{N} \rightarrow \mathbb{B}$  seen as an interval extension of  $f : \mathbb{R} \rightarrow \mathbb{S}$  is tight if whenever  $(f x) \downarrow$ , there (classically) exists an interval containing  $x$  for which  $\hat{f}$  indicates that  $\downarrow$  is the only valid answer on that interval:

$$\text{tight } \hat{f} := \Pi(x : \mathbb{R}). (f x) \downarrow \rightarrow \exists(d : \mathbb{D}). \exists(n : \mathbb{N}). |x - d| < 2^{-n} \wedge \hat{f}(d, n) = \text{true}$$

Our continuity principle is the following axiom saying that for any map  $f : \mathbb{R} \rightarrow \mathbb{S}$ , there nondeterministically exists an interval extension of it that is tight:

$$\Pi(f : \mathbb{R} \rightarrow \mathbb{S}). \text{M}\Sigma(\hat{f} : \mathbb{D} \times \mathbb{N} \rightarrow \text{bool}). \text{incl } \hat{f} \wedge \text{tight } \hat{f}.$$

It is realized by the trivial operation that given  $f$  extracts the realizer  $\hat{f}$  of  $f$  assuming that the underlying exact real computation is done via dealing real numbers by converging sequences of intervals with dyadic rational endpoints. Since a single function admits several different realizers, the procedure of extracting realizers is nondeterministic.

From the above, we can derive the more standard form of the continuity principle which says that there exists a modulus function:

$$\Pi(f : \mathbb{R} \rightarrow \mathbb{S}). \Pi(x : \mathbb{R}). (f x) \downarrow \rightarrow \text{M}\Sigma(n : \mathbb{N}). \Pi(y : \mathbb{N}). |x - y| < 2^{-n} \rightarrow (f y) \downarrow.$$

To derive this version of the principle, assume that we are given  $x$  with  $f x \downarrow$ . Due to the continuity axiom, we further have nondeterministically a tight interval extension  $\hat{f}$ . From the tightness, we then know that there (classically) exist  $d, n$  such that  $|d - x| < 2^{-n}$  and  $\hat{f}(d, n) = \text{true}$ . Though  $\hat{f}$  is Boolean-valued, we can post-compose the embedding from  $\mathbb{B}$  to  $\mathbb{S}$  and use the nondeterministic countable choice principle in Equation (1) to nondeterministically find such  $d$  and  $n$ . We then (again nondeterministically) choose some  $m$  such that  $2^{-m} < 2^{-n} - |x - d|$ . As for any  $y$  with  $|x - y| < 2^{-m}$ ,  $|y - d| < 2^{-n}$  holds, it is guaranteed that  $f y \downarrow$  by the properties of the interval extension. Thus, any such  $m$  fulfills the condition.

Further, we can derive a continuity principle for real valued functions  $f : \mathbb{R} \rightarrow \mathbb{R}$ :

$$\Pi(x : \mathbb{R}). \Pi(m : \mathbb{N}). \text{M}\Sigma(n : \mathbb{N}). \Pi(y : \mathbb{N}). |x - y| < 2^{-n} \rightarrow |(f x) - (f y)| < 2^{-m}$$

To prove that this continuity principle holds, take any  $f : \mathbb{R} \rightarrow \mathbb{R}$ ,  $x : \mathbb{R}$ , and  $m : \mathbb{N}$ , and define a function  $g_{x,m} : \mathbb{R} \rightarrow \mathbb{S}$  by  $g_{x,m} := \lambda(y : \mathbb{R}). |(f x) - (f y)| < 2^{-m}$ . Here, we let  $(x < y)$  be the embedding of the Kleenean valued comparison to Sierpinski space, i.e. such that  $(x < y) \downarrow$  if and only if  $x < y$  holds. The continuity principle on Sierpinski valued functions yields

$$|(f x) - (f x)| < 2^{-m} \rightarrow \text{M}\Sigma(n : \mathbb{N}). \Pi(y : \mathbb{R}). |x - y| < 2^{-n} \rightarrow |(f x) - (f y)| < 2^{-m}.$$

As  $|(f x) - (f x)| < 2^{-m}$  is trivially true, we get the desired bound.

As a last application, consider any discrete type  $X$  having decidable equality  $\Pi(x, y : X). (x = y) + (x \neq y)$ . We can then prove that any  $f : \mathbb{R} \rightarrow X$  is constant:  $\Pi(x : \mathbb{R}). \Pi(y : \mathbb{R}). f x = f y$ . Similarly as above, assuming  $x : \mathbb{R}$ , we define a function  $g_x$  where  $(g_x y)$  is defined if  $(f x) = (f y)$  and undefined if  $(f x) \neq (f y)$ , constructed based on the decidability of elements of  $X$ . Again, applying the continuity principle, we get

$$\text{M}\Sigma(n : \mathbb{N}). \Pi(y : \mathbb{R}). |x - y| < 2^{-n} \rightarrow (f x) = (f y) .$$

Since we can prove that any locally constant function from  $\mathbb{R}$  is constant in the classical part of our system, we can conclude that  $f$  is constant.

All results can be easily generalized to arbitrary Euclidean space  $\mathbb{R}^m$  by replacing the absolute value with the maximum norm and proofs are contained in our formal development.

## 4 Subsets

For any type  $X$ , we first express subsets classically as predicates  $X \rightarrow \text{Prop}$ . That is, we set

$$\text{csubset}(X) := X \rightarrow \text{Prop}$$

and define classical operations on classical subsets such as  $\in, \cup, \cap, \subseteq, \dots$  in the obvious way. For example,  $(x \in S) := S x$ ,  $S \cap T := \lambda(x : X). x \in S \wedge x \in T$ , and so on.

Our goal is to assign computational content to classically defined subsets which in turn allows to extract programs that perform operations on these sets. In computable analysis, computational content is given by defining representations for certain spaces of subsets [22, 8] and perform computations on these representations. This can be done in a very general setting without restricting to specific spaces, although the computational procedures are usually extremely inefficient and not useful for practical applications.

While we are mostly interested in subsets of Euclidean spaces for which we can specialize the theorems and get more efficient algorithms, the general results are still interesting mathematically and can be used for basic facts that are not meant to be used computationally.

Classically, a subset  $A \subseteq X$  of a topological space  $X$  is open iff its characteristic function  $\chi_A : X \rightarrow \mathbb{S}$  is continuous. We thus identify open sets with their characteristic function:

$$\text{open } A := \Sigma(f : X \rightarrow \mathbb{S}). \Pi(x : X). (f x) \downarrow \leftrightarrow x \in A .$$

Similarly, we define closed sets as their complement

$$\text{closed } A := \Sigma(f : X \rightarrow \mathbb{S}). \Pi(x : X). (f x) \downarrow \leftrightarrow x \notin A .$$

Computationally for an open set we can verify (i.e. semi-decide) if a point is contained in the set and for a closed set we can verify if a point is outside.

We state some simple properties of open and closed sets. We omit the proofs here, but pen and paper proofs can e.g. be found in [22] and full formal proofs can be found in our Coq development.

1. If  $(A_i)_{i \in \mathbb{N}}$  are open then so is the countable union  $\bigcup A_i$ .
2. If  $A$  and  $B$  are open then so is their finite intersection  $A \cap B$ .
3. If  $A$  and  $B$  are closed then so is their finite union  $A \cup B$ .
4. If  $(A_i)_{i \in \mathbb{N}}$  are closed then so is the countable intersection  $\bigcap A_i$ .

Other classes of subsets that are often considered in computable analysis are the compact and overt subsets. (Computably) compact subsets are those for which we can effectively verify that an open set covers them. That is, a subset  $A \subseteq X$  is compact iff

$$\Sigma(f : \text{open } X \rightarrow \mathbb{S}). \Pi(U : \text{open } X). (f U) \downarrow \leftrightarrow A \subseteq U .$$

Overtness is a dual notion to compactness. For a (computably) overt subset, we can effectively verify if an open set touches it. That is, a subset  $A \subseteq X$  is overt iff

$$\Sigma(f : \text{open } X \rightarrow \mathbb{S}). \Pi(U : \text{open } X). (f U) \downarrow \leftrightarrow A \cap U \neq \emptyset.$$

The notion of overtness is less known to classical mathematicians as it does not carry any information outside a constructive or computational context (i.e. any set is classically overt).

Another way to look at these notions is that for a compact subset  $A$  it can be verified that a semidecidable property  $P$  holds for each point, i.e. if  $\forall x \in A, P x$  and for an overt subset it can be verified that it holds for at least one point, i.e. if  $\exists x \in A, P x$ .

Similar as for open and closed sets, we prove several properties of compact and overt sets in our Coq development but omit the details here for space reasons.

## 5 Subsets of Euclidean Space

Using basic properties of the Kleenean and Sierpinski spaces, we get short and elegant proofs for simple properties of open sets. However, from the point of view of doing actual computations, using Sierpinski valued functions to represent basic objects is often far from optimal and programs extracted from the proofs are inefficient. As we are interested in extracting exact real computation programs, our main focus is on subsets of Euclidean spaces. Most statements in this section can be proven in a much more general setting (and our Coq development contains some of them). However, the proofs in this section are optimized to encode efficient algorithms by making use of particular properties of the representation.

### 5.1 Characterization of Euclidean Subsets

We prove the following characterization for subsets  $A \subseteq \mathbb{R}^m$  of Euclidean space:

$$\begin{aligned} \text{open } A &\leftrightarrow \text{M}\Sigma(F : \mathbb{N} \rightarrow \mathbb{R}^m \times \mathbb{R}). \\ &\quad \Pi(n : \mathbb{N}). \text{B}((F n)) \subseteq A \\ &\quad \wedge \quad \Pi(x : \mathbb{R}^m). x \in A \rightarrow \exists(n : \mathbb{N}). x \in \text{B}((F n)). \end{aligned}$$

Here,  $\text{B}(x, r)$  encodes a ball with radius  $r$  around  $x$ . That is, a subset  $A \subseteq \mathbb{R}^m$  is open iff we can find a sequence of (possibly empty) balls, all contained in  $A$ , that eventually cover all  $A$ .

To show the equivalence let us call the statement on the right of the equivalence Euclidean open. Thus, first assume that  $A$  is open in the sense of the previous section, that is, there is a function  $f : \mathbb{R}^m \rightarrow \mathbb{S}$  such that  $(f x) \downarrow$  iff  $x \in A$ . We can enumerate all possible inputs  $\mathbb{D}^d \times \mathbb{N}$  and define a sequence of balls by choosing  $\text{B}(d, 2^{-n})$  whenever  $(\hat{f} d n) = \text{True}$ . By the properties of the interval extension, any such ball is contained in  $A$  and for any  $x \in A$  we will find at least one ball that contains  $x$ .

Now assume  $A$  is Euclidean open. For any  $x, y, r$ , testing if  $x \in \text{B}(y, r)$  is semidecidable as we only need to check if  $d(x, y) < r$ . Thus, given  $x : \mathbb{R}$  we define a sequence  $s : \mathbb{N} \rightarrow \mathbb{S}$  such that  $(s n) \downarrow$  iff  $x \in \text{B}((F n))$ . Then  $\exists(n : \mathbb{N}). (s n) \downarrow \leftrightarrow x \in A$ , i.e., we need to check if there is at least one element in the sequence that is defined, which is again semidecidable.

For practical purposes, we are mostly interested in Euclidean subsets that are both compact and overt. Such sets (and some variations) have been considered in constructive mathematics and computable analysis under different names, sometimes using the same terminology for slightly different concepts. In Bishop's constructive mathematics, a set is compact if it is closed and totally bounded [10]. Here, totally bounded means that it can be covered by finitely many subsets of fixed size. Brattka and Weihrauch [4] define a representation  $\nu_{\text{min-cover}}$  for compact sets based on such coverings.

We only consider subsets of Euclidean space that are classically closed and characterize their subsets in terms of “drawings” with arbitrary precision, meaning that for each  $n$  we can generate a picture of  $A$  in terms of “pixels” (i.e. small boxes or closed balls) of size  $2^{-n}$ . It turns out that subsets for which such drawings exist are precisely those that are both compact and overt. It can further shown that they coincide with the *located* subsets, i.e., the subsets for which the distance of the set and a real number is computable (cf. [27, §12] and [24]). We thus use the term *located* for these subsets. We can define a closed ball  $B(c, r) \subseteq \mathbb{R}^m$  with center  $c$  and radius  $r$  (w.r.t. the maximum norm) simply as the tuple  $(c, r) : \mathbb{R}^m \times \mathbb{R}$ . We denote the type of all such balls by  $B$  and use  $[B]$  to denote finite lists of balls. For any  $b : B$  we further let  $\|b\|$  denote its radius. We then formally define the located sets by

$$\begin{aligned} \text{located } A &:\equiv \Pi(n : \mathbb{N}). \Sigma(L : [B]). \\ &\quad \Pi(b \in L). \|b\| \leq 2^{-n} \quad (\text{fast convergence}) \\ &\quad \wedge \Pi(b \in L). b \cap A \neq \emptyset \quad (\text{intersection}) \\ &\quad \wedge \bigcup_{b \in L} b \supseteq A \quad (\text{cover}). \end{aligned}$$

Thus, a set is located if for each  $n$  we get a finite list of closed balls, each with radius at most  $2^{-n}$  so that their union covers the whole set  $A$  and such that each ball intersects  $A$ . Note that we need to restrict to classically closed subsets as the coverings only define a set up to its closure. The empty set is located as we allow closed balls to be empty (encoded by a negative radius).

We call the set defined by the union of the elements of the  $n$ -th list of the sequence the  $n$ -th approximation or the  $n$ -th covering of  $A$  and denote it by  $A_n$ .

For  $A \subseteq \mathbb{R}^m$  and  $x \in \mathbb{R}^m$ , let us write  $d(x, A)$  for  $\inf_{y \in A} \|x - y\|$ .

The Hausdorff distance  $d_H(A, B)$  of two sets is defined by

$$d_H(X, Y) = \max \left\{ \sup_{x \in X} d(x, Y), \sup_{y \in Y} d(X, y) \right\}.$$

Note that the set defined by the  $n$ -th approximation has Hausdorff distance at most  $2^{-n}$  to the set that is approximated. In fact, the other direction also holds, i.e. we can show

$$\text{located } A \leftrightarrow \Pi(n : \mathbb{N}). \Sigma(L : [B]). d_H(A, \bigcup_{b \in L} b) \leq 2^{-n}.$$

To see this, assume for each  $n \in \mathbb{N}$  we can get a list of balls  $L$  such that  $d_H(A, \bigcup_{b \in L} b) \leq 2^{-n}$ . We take the  $(n + 1)$ -st list and double the radius of each of the balls. This guarantees that the resulting set covers  $A$  while still being small enough.

Note that center and radius of the balls can be arbitrary real numbers, as in the context of exact real computation it is more convenient to work with real numbers than with rationals. However, decidable equality on balls can be useful. Restricting to only dyadic rationals for center and radius does not make a difference as we can approximate real numbers arbitrarily well by dyadic rationals and thus can convert any real covering to a dyadic covering.

It can be shown that a set is located if and only if it is compact and overt. Although a formal proof of the equivalence could be used to prove that some of the subset operations preserve locatedness, recall that the main goal of our project is to extract efficient programs from proofs. It is thus reasonable to have different proofs using the more efficient representation, and a formal proof of the equivalence is less important. While the algorithmic content of the proofs is rather simple, showing correctness of the procedures needs some facts from classical analysis complicating the formalization of the proof. We therefore only give a pen and paper proof of the equivalence in Appendix A and state the fact as a meta-theorem.

## 5.2 Operations on located subsets

Let us first show that we can effectively get the distance of a point and a located set. For any nonempty located sets we can compute the distance function. That is,

$$\text{located } A \wedge A \neq \emptyset \rightarrow \Pi(x : \mathbb{R}^m). \Sigma(r : \mathbb{R}). r = d(x, A).$$

For any ball  $b = B(c, r)$ , the distance  $d(b, x)$  is given by  $\max\{0, \|x - c\| - r\}$ . We choose the  $n$ -th approximation and take the minimum over all  $d(b, x)$ . This gives a  $2^{-n}$  approximation of  $d(A, x)$ . Taking the limit of all the approximations for each  $n$  thus corresponds to  $d(A, x)$ .

Let us next show that several standard operations on located subsets preserve locatedness. Although the results already follow from located sets being both compact and overt, our proofs encode more efficient algorithms, making use of the representation of located sets. The extracted programs can be used as a small calculus to combine subsets using set operations.

The properties stated below mostly can be shown by directly applying the operations on the coverings. We thus state them without proof.

1. For located sets  $A, B$  their union  $A \cup B$  is located.
2. For a located set  $A$  and any  $\lambda > 0$ , the scaled set  $\lambda A := \{\lambda x : x \in A\}$  is located.
3. For a located set  $A$  and any  $v : \mathbb{R}^m$ , the set  $A + v := \{x + v : x \in A\}$  is located.

Using the continuity principle on real numbers we can generalize the above to images of arbitrary real functions. For any set  $A \subseteq \mathbb{R}^m$ , and function  $f : \mathbb{R}^m \rightarrow \mathbb{R}^k$  let us define the image  $f[A] \subseteq \mathbb{R}^k$  by  $y \in f[A] :\leftrightarrow \exists(x : A). f(x) = y$ . Then we can show that the image of located sets is again located:

$$\Pi(A \subseteq \mathbb{R}^m). \Pi(f : \mathbb{R}^m \rightarrow \mathbb{R}^k). \text{located } A \rightarrow \text{located } f[A]$$

To show this, for each  $n \in \mathbb{N}$ , we need to find a covering of  $f[A]$  with balls with radius at most  $2^{-n}$ . We can use the continuity principle on real functions to define an extension  $\hat{f}$  that maps balls  $b \subseteq \mathbb{R}^m$  to balls (with possibly infinite radius)  $b' \subseteq \mathbb{R}^k$  such that  $f[b] \subseteq b'$ . We can further show that the radius of  $\hat{f}(b)$  goes to zero when the radius of  $b$  goes to zero. Thus we generate all coverings, apply  $\hat{f}$  to each of the balls of the coverings and check if the radius of each of them is bounded by  $2^{-n}$  which we know will eventually happen. While generating images under arbitrary functions is very powerful, the procedure is obviously quite inefficient and thus generating sets using the above specialized operations is preferred.

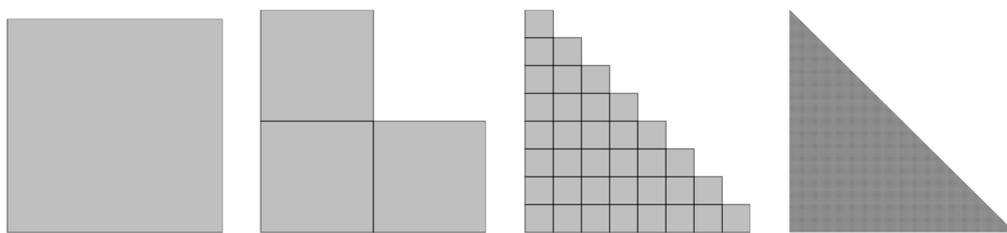
Another very useful operation that lets us easily generate more complicated sets from simple ones is the limit operation. Here, by limit operation we mean that we are given a sequence of located sets which converge to another set in terms of Hausdorff distance. That is, a set  $K$  is defined to be the limit of a sequence of located sets if

$$\text{isLim } K := \Pi(n : \mathbb{N}). \Sigma(X : \text{located } \mathbb{R}^m). d_H(X, K) \leq 2^{-n}$$

holds. We can show that the limit of a sequence of located sets is again located, i.e.

$$\text{isLim } K \rightarrow \text{located } K.$$

To see this, let  $X_i$  denote the  $i$ -th set in the sequence and  $X_{i,j}$  denote its  $j$ -th covering. As  $d_H(X_i, K) \leq 2^{-i}$  and  $d(X_{i,j}, X_i) \leq 2^{-j}$  it suffices to take  $X_{n+1, n+1}$  and double the radius of each of the balls to guarantee that the resulting set covers  $K$ .



■ **Figure 1** Approximations of the triangle.

## 6 Examples

From a proof that a set is located in our Coq implementation, we can extract an AERN program that computes the coverings. We implemented the examples below in Coq and extracted programs to generate for each  $n$  a finite list of real numbers, encoding the center and radii of each ball in the  $n$ -th covering. We can then use AERN to give us arbitrarily exact rational approximations of these real numbers, which we in turn can output and visualize by drawing the approximate boxes.

### 6.1 A Simple Triangle

Let us start with a simple example. We define a triangle  $T \subseteq \mathbb{R}^2$  by

$$(x, y) \in T \leftrightarrow x \geq 0 \wedge y \geq 0 \wedge x + y \leq 1.$$

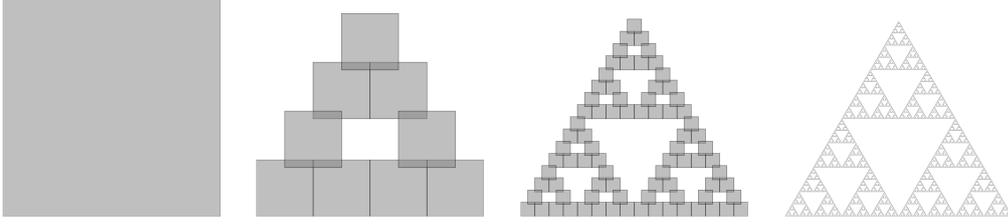
To show that the set is located, let  $b_{i,j,n} := B((\frac{2i+1}{2^{n+1}}, \frac{2j+1}{2^{n+1}}), 2^{-n})$ . We define the  $n$ -th covering as the list  $L$  containing all  $b_{i,j,n}$  with  $i + j < 2^n$ . Figure 1 shows the coverings defined by this procedure. All of the balls have radius  $2^{-n}$  by definition and each ball intersects the triangle as  $i + j < 2^n$  implies  $\frac{2i+1+2j+1}{2^{n+1}} \leq 1$ . Each point of the triangle is contained in one of the  $b_{i,j,n}$  as for any  $x \in [0, 1]$  and  $n \in \mathbb{N}$  we can find some  $k \in \mathbb{N}$  such that  $2^{-n}k \leq x < 2^{-n}(k+1)$ . We can thus find such coordinates  $(i, j)$  for a point  $(x, y) \in T$  and as  $x + y \leq 1$  it follows that  $i + j < 2^n$  as claimed.

### 6.2 Fractals

As a more interesting application, let us look at a procedure to generate certified drawing of fractals. We consider simple self-similar fractals generated by iterated function systems (IFPs) without rotation. For simplicity we only consider fractals that are contained in the unit cube  $[0, 1]^m$ . For a finite set of points  $D := (d_1, d_2, \dots, d_k) \subseteq [0, 1]^m$  we define a classical subset  $\mathcal{I}(D) \subseteq \mathbb{R}^m$  as the smallest subset such that

1.  $D \subseteq \mathcal{I}(D)$ , and
2.  $\Pi(x : \mathcal{I}(D)). \Pi(d : D). \frac{x+d}{2} \in \mathcal{I}(D)$ .

Classically it can be shown that there is a unique compact subset with these properties. We show that any set defined in this way is located. To do so, we need to define a sequence of coverings of the set. We start with the first covering  $L_0$  being the list containing only the unit disc. As each  $d \in D$  is contained in the unit disc, the intersection and covering properties hold. We then recursively define the covering  $L_{n+1}$  from the previous covering  $L_n$  by following the construction rule, i.e., by making for each  $d \in D$  a copy of the previous covering where the center of each ball is moved halfway towards  $d$  and has half the radius.



■ **Figure 2** Approximations of the Sierpinski triangle.

Obviously, each of the balls in  $L_{n+1}$  has radius  $2^{-(n+1)}$ . To show that this procedure preserves the intersection property, let  $b \in L_{n+1}$ . Then there is some  $b' \in L_n$  and  $d \in D$  such that  $x \in b$  iff there is an  $x' \in b'$  with  $x = \frac{x'+d}{2}$ . As  $L_n$  has the intersection property, there is some  $x' \in b'$  with  $x' \in \mathcal{I}(D)$ . By definition of  $\mathcal{I}(D)$ , then  $\frac{x'+d}{2} \in \mathcal{I}(D)$ , i.e.,  $b \cap \mathcal{I}(D) \neq \emptyset$ . Similarly, to show that the covering property is preserved, assume  $x \in \mathcal{I}(D)$ . Then by definition either  $x = d$  for some  $d \in D$  or there is some  $x' \in \mathcal{I}(D)$  and some  $d \in D$  such that  $x = \frac{x'+d}{2}$ . Any  $d \in D$  is contained in  $L_{n+1}$  as  $d \in L_n$  and  $d = \frac{d+d}{2}$ . In the other case, there is some  $b \in L_n$  such that  $x' \in b$  as  $L_n$  is a covering. But then by definition of the procedure, there is some  $b' \in L_{n+1}$  that contains  $x$ .

A more elegant way to prove the locatedness of  $\mathcal{I}(D)$  is using the limit operation. We can define a sequence of located sets  $(T_i)_{i \in \mathbb{N}}$  by letting  $T_0$  be the unit disc and then applying the iteration given by the IFP. It is not hard to show that each  $T_i$  has Hausdorff distance at most  $2^{-i}$  to the fractal and thus applying the limit operation suffices. However the extracted program is slightly less efficient than the direct encoding as the limit uses  $T_{n+1}$  to get the  $n$ -th approximations and increases the radius leading to coverings where balls overlap.

As an application of the result, consider the Sierpinski triangle which is defined as the two dimensional subset  $\mathcal{I}(D)$  with  $D = \{(-1, -1), (1, -1), (0, \sqrt{3} - 1)\}$ .

From the above we get a proof that the set is located, from which we can extract a Haskell program computing the coverings. Figure 2 is a plot of the output of the extracted program for increasing  $n$ . To generate the plot, AERN has been used to approximate the real numbers for center and radius in the  $n$ -th approximation with error bounded by  $2^{-n}$ . The overlap at the top and bottom of the boxes is due to the approximation of the real number  $\sqrt{3} - 1$ .

## 7 Conclusion and future work

We considered procedures to work with different classes of subsets in a computational setting. Currently, we only consider simple operations on subsets to build new subsets and extract programs for certified drawings up to any desired resolution. Producing coverings yields a representation that is easy to manipulate computationally and efficient if our goal is to draw the whole set or do other operations that require global knowledge. However, naturally the size of the list of coverings grows exponentially when increasing the precision. One might argue that high precision approximations are more useful locally, i.e. instead of getting a picture of the whole set one might want to zoom into a small part of the set and draw it with arbitrarily high precision.

Instead of outputting the whole set, the list can be replaced by a nondeterministic test function, that tests whether a ball is close enough to the set or not:

$$\Pi(x : \mathbb{R}^d). \Pi(r : \mathbb{R}). M((d(x, A) < r) + (d(x, A) > 2r)).$$

That is, given a ball with center  $x$  and radius  $r$ , the function tells us if the ball intersects the set or not, but it is allowed to slightly over-approximate the set, i.e., both answers are possible if a ball with twice the radius intersects the set. Similar representations are often used in complexity theory (e.g. [6, 7]). In future work, we plan to include such representations in our development and show the equivalence to the covering representation for located sets.

More interesting operations can be achieved from computing images of arbitrary functions. Currently, we only have a simple implementation using the continuity principle on real numbers which is not efficient enough to be used in actual applications. Similar to the representations of Euclidean subsets, we can think of more efficient representations for the space of real functions. One possibility that comes to mind is using polynomial models such as Taylor models [18]. This would also allow for extensions to more advanced operations like integration and solution operators for ordinary differential equations [12, 21].

---

## References

- 1 A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A.L. Sangiovanni-Vincentelli. Ariadne: a Framework for Reachability Analysis of Hybrid Automata. In *Proc. 17th Int. Symp. on Mathematical Theory of Networks and Systems*, Kyoto, 2006.
- 2 Ulrich Berger and Hideki Tsuiki. Intuitionistic fixed point logic. *Ann. Pure Appl. Log.*, 172(3):102903, 2021. doi:10.1016/j.apal.2020.102903.
- 3 Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016. URL: <http://hal.inria.fr/hal-00806920>.
- 4 Vasco Brattka and Klaus Weihrauch. Computability on subsets of euclidean space i: closed and compact subsets. *Theoretical Computer Science*, 219(1):65–93, 1999. doi:10.1016/S0304-3975(98)00284-9.
- 5 Franz Brauße, Pieter Collins, and Martin Ziegler. Computer science for continuous data. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 62–82, Cham, 2022. Springer International Publishing.
- 6 Mark Braverman. On the complexity of real functions. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 155–164. IEEE, 2005.
- 7 Mark Braverman and Michael Yampolsky. *Computability of Julia sets*, volume 23. Springer, 2009.
- 8 Pieter Collins. Computable analysis with applications to dynamic systems. *Mathematical Structures in Computer Science*, 30(2):173–233, 2020. doi:10.1017/S096012952000002X.
- 9 Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.
- 10 Hannes Diener. *Compactness under constructive scrutiny*. PhD thesis, University of Canterbury. Mathematics and Statistics, 2008.
- 11 Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 427–441, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- 12 Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ODE solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.
- 13 Michal Konečný, Sewon Park, and Holger Thies. Certified computation of nondeterministic limits. In *NASA Formal Methods Symposium*, pages 771–789. Springer, 2022.
- 14 Michal Konečný, Sewon Park, and Holger Thies. Extracting efficient exact real number computation from proofs in constructive type theory. *arXiv preprint*, 2022. arXiv:2202.00891.

- 15 Michal Konečný. aern2-real: A Haskell library for exact real number computation. <https://hackage.haskell.org/package/aern2-real>, 2021.
- 16 Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical computer science*, 38:35–53, 1985.
- 17 Horst Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, 5(3):321–324, 1977. doi:10.1016/0304-3975(77)90048-2.
- 18 Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 6:239–316, 2003.
- 19 Valérie Ménessier-Morain. Arbitrary precision real arithmetic: design and algorithms. *J. Log. Algebr. Program.*, 64(1):13–39, 2005. doi:10.1016/j.jlap.2004.07.003.
- 20 Norbert Th Müller. The iRRAM: Exact arithmetic in C++. In *International Workshop on Computability and Complexity in Analysis*, pages 222–252. Springer, 2000.
- 21 Nedialko S Nedialkov. Interval tools for odes and daes. In *12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2006)*, pages 4–4. IEEE, 2006.
- 22 Arno Pauly. On the topological aspects of the theory of represented spaces. *Comput.*, 5:159–180, 2016.
- 23 R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33–48, 1984. doi:10.1017/S0305004100061284.
- 24 Bas Spitters. Locatedness and overt sublocales. *Annals of Pure and Applied Logic*, 162(1):36–54, 2010.
- 25 Dieter Spreen and Ulrich Berger. Computing with Infinite Objects: the Gray Code Case. *Logical Methods in Computer Science*, Volume 19, Issue 3, July 2023. URL: <https://lmcs.episciences.org/11550>.
- 26 Florian Steinberg, Laurent Theys, and Holger Thies. Computable analysis and notions of continuity in Coq. *Logical Methods in Computer Science*, Volume 17, Issue 2, May 2021. URL: <https://lmcs.episciences.org/7478>.
- 27 Paul Taylor. A lambda calculus for real analysis. *Journal of Logic and Analysis*, 2, 2010.
- 28 K. Weihrauch. *Computable analysis*. Springer, Berlin, 2000.

## A Equivalence of representations

We prove that a classically closed set is located if and only if it is both compact and overt. Since the *if* part is straightforward, let us focus on the *only if* direction.

Let us assume that  $A$  is compact and overt and show that  $A$  is located as well. We need to find for any natural  $n$  a finite set of balls of radius at most  $2^{-n}$ , each intersecting  $A$ , and collectively covering  $A$ . We will also assume  $n \geq 1$  and assign to  $n = 0$  the same set of balls as for  $n = 1$ .

As  $A$  is computationally compact, we can find a ball containing  $A$ . Without loss of generality, assume  $A \subseteq B(0, 1)$ .

Let  $J$  be the set of all  $d$ -dimensional integer vectors  $j$  whose integer components  $j_i$  satisfy  $-m \leq j_i \leq m$  where  $m = 3 \cdot 2^{n-1}$ . The points  $c_j = j/m$  for  $j \in J$  form a uniform grid over the (max-norm) unit ball  $B(0, 1)$ . Therefore the union of balls of radius  $r = 1/2m$  around  $c_j$  covers  $B(0, 1)$  and therefore  $A$ :

$$A \subseteq B(0, 1) \subset \bigcup_{j \in J} \overline{B}(c_j, r) \tag{2}$$

We will apply compact  $A$  and overt  $A$  on open and closed sets, respectively, derived from the above cover of  $A$ , except that we increase the radius to  $3r = 3/2m = 2^{-n}$ . This will make the cover sufficiently redundant to compensate for potential non-termination of the compactness and overtness applications on specific sets.

For each  $j \in J$ , let

$$\begin{aligned} \text{disjoint}_j &= \text{compact } A \ (A \cap \overline{B}(c_j, 3r)) \\ \text{intersects}_j &= \text{overt } A \ (A \cap B(c_j, 3r)) \\ \text{haveinfo}_j &= \text{empty}_j \sqcup \text{nonempty}_j \end{aligned}$$

Note that as  $\text{disjoint}_j, \text{intersects}_j, \text{haveinfo}_j \in \mathcal{S}$ , their computation may not terminate. Moreover,  $\text{haveinfo}_j$  terminates iff either of the other two terminate.

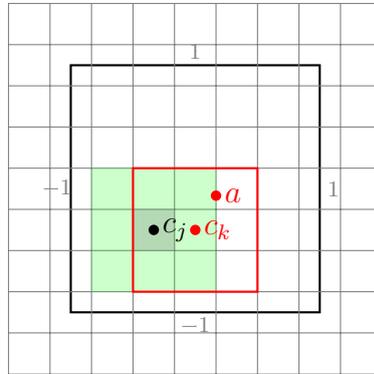
We compute all  $\text{haveinfo}_j$  computations in parallel until enough of them succeed to cover  $A$ , more precisely, until for the set of terminated indices  $J_T \subseteq J$ , it holds:

$$B(0, 1) \subseteq \bigcup_{j \in J_T} \overline{B}(c_j, 3r) \quad (3)$$

Once Equation (3) holds, the set  $\{\overline{B}(c_j, 3r) \mid j \in J_T \wedge \text{intersects}_j\}$  is the desired covering, namely, the balls have sufficiently small radius, each of them intersects  $A$ , and their union covers  $A$  thanks to Equation (3) and the definition of  $\text{disjoint}_j$ , which means that we omit only balls that are disjoint from  $A$ .

It remains to prove that Equation (3) must hold in finite time. This will follow from Equation (2) when we show that for each  $j \in J$  the ball  $\overline{B}(c_j, r)$  is covered by some  $\overline{B}(c_k, 3r)$  such that  $\text{haveinfo}_k$  terminates. Note that the ball  $\overline{B}(c_j, r)$  is covered by  $\overline{B}(c_k, 3r)$  for all  $k \in J$  with  $\|j - k\| \leq 1$ , i.e., the “neighbouring” indices in addition to  $j$  itself.

Now, if  $\text{haveinfo}_j$  does not terminate, i.e., if both  $\text{empty}_j$  and  $\text{nonempty}_j$  do not terminate, the ball  $\overline{B}(c_j, 3r)$  is not disjoint from  $A$  but its interior is disjoint from  $A$ . This means that there is  $a \in A$  with  $\|c_j - a\| = 3r$  as illustrated in Figure 3.



■ **Figure 3**  $\overline{B}(c_j, r)$  covered by  $\overline{B}(c_k, 3r)$ ;  $n = 1, m = 3, r = 1/6, j = (-1, -1)$ .

Let us define the vector  $k$  component-wise as follows:

$$k_i = \begin{cases} j_i + 1 & a_i - (c_j)_i = 3r \\ j_i - 1 & a_i - (c_j)_i = -3r \\ j_i & |a_i - (c_j)_i| < 3r \end{cases} \quad (4)$$

Note that  $k$  is a “neighbour” of index  $j$ . Now,  $k \in J$  because if it was not, some component  $k_i$  would be outside the interval  $[-m, m]$ , which would mean that either  $a_i = 1 + 3r$  or  $a_i = -1 - 3r$ , which would contradict  $A \subseteq B(0, 1)$ .

## 59:16 Formalizing Hyperspaces for Extracting Efficient Exact Real Computation

Using the same three cases as in Equation (4), we get:

$$|a_i - (c_k)_i| = \begin{cases} |a_i - (c_j)_i - 1/m| = 3r - 1/m < 3r & k_i = j_i + 1 \\ |a_i - (c_j)_i + 1/m| = |-3r + 1/m| < 3r & k_i = j_i - 1 \\ |a_i - (c_j)_i| < 3r & k_i = j_i \end{cases} \quad (5)$$

In all three cases we have  $|(c_k)_i - a_i| < 3r$ . Over all  $i$ , we get  $a \in B(c_k, 3r)$ .

Finally,  $a \in B(c_k, r)$  implies that  $\text{nonempty}_k$  terminates, and therefore  $\overline{B}(c_j, r)$  is covered by  $\overline{B}(c_k, 3r)$  in Equation (3) as illustrated in Figure 3.

# Set Semantics for Asynchronous TeamLTL: Expressivity and Complexity

Juha Kontinen  

University of Helsinki, Finland

Max Sandström  

University of Sheffield, UK

University of Helsinki, Finland

Jonni Virtema  

University of Sheffield, UK

University of Helsinki, Finland

---

## Abstract

We introduce and develop a set-based semantics for asynchronous TeamLTL. We consider two canonical logics in this setting: the extensions of TeamLTL by the Boolean disjunction and by the Boolean negation. We relate the new semantics with the original semantics based on multisets and establish one of the first positive complexity theoretic results in the temporal team semantics setting. In particular we show that both logics enjoy normal forms that can be utilised to obtain results related to expressivity and complexity (decidability) of the new logics.

**2012 ACM Subject Classification** Theory of computation → Logic and verification

**Keywords and phrases** Hyperproperties, Linear Temporal Logic, Team Semantics

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.60

**Related Version** *Full Version:* <https://arxiv.org/abs/2304.10915> [13]

**Funding** *Juha Kontinen:* Supported by the Academy of Finland grant 345634.

*Max Sandström:* Supported by the Academy of Finland grant 322795.

*Jonni Virtema:* Supported by the Academy of Finland grant 345634 and the DFG grant VI 1045/1-1.

## 1 Introduction

Linear temporal logic (LTL) is one of the most prominent logics for the specification and verification of reactive and concurrent systems. The core idea in model checking, as introduced in 1977 by Amir Pnueli [22], is to specify the correctness of a program as a set of infinite sequences, called traces, which define the acceptable executions of the system. In LTL-model checking one is concerned with trace sets that are definable by an LTL-formula. Ordinary LTL and its progeny are well suited for specification and verification of *trace properties*. These are properties of systems that can be checked by going through all executions of the system in isolation. A canonical example here is *termination*; a system terminates if each run of the system terminates. However not all properties of interest are trace properties. Many properties that are of prime interest, e.g., in information flow security, require a richer framework. The term *hyperproperty* was coined by Clarkson and Schneider [3] to refer to properties which relate multiple execution traces. A canonical example is *bounded termination*; one cannot check whether a system terminates in bounded time by only checking traces in isolation. Checking hyperproperties is vital in information flow security where dependencies between secret inputs and publicly observable outputs of a system are considered potential security violations. Commonly known properties of that type are noninterference [24, 20] and observational determinism [30]. Hyperproperties are not limited to the area of information flow control; e.g., distributivity and other system properties like fault tolerance can be expressed as hyperproperties [5].



© Juha Kontinen, Max Sandström, and Jonni Virtema;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 60; pp. 60:1–60:14

Leibniz International Proceedings in Informatics



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

During the past decade, the need for being able to formally specify hyperproperties has led to the creation of families of new logics for this purpose, since LTL and other established temporal logics can only specify trace properties. The two main families of the new logics are the so-called *hyperlogics* and logics that adopt *team semantics*. In the former approach temporal logics such as LTL, computation tree logic (CTL), and quantified propositional temporal logic (QPTL) are extended with explicit trace and path quantification, resulting in logics like HyperLTL [2], HyperCTL\* [2], and HyperQPTL [23, 4]. The latter approach (which we adopt here) is to lift the semantics of temporal logics to sets of traces directly by adopting team semantics yielding logics such as TeamLTL [15, 7] and TeamCTL [14, 7].

Krebs et al. [15] introduced two versions of LTL with team semantics: a synchronous semantics and an asynchronous variant that differ on how the evolution of time is linked between computation traces when temporal operators are evaluated. In the synchronous semantics time proceeds in lock-step, while in the asynchronous variant time proceeds independently on each trace. For example the formula “F terminate” (here F denotes the future-operator and “terminate” is a proposition depicting that a trace has terminated) defines the hyperproperty “bounded termination” under synchronous semantics, while it expresses the trace property “termination” under asynchronous semantics. The elegant definition of bounded termination exemplifies one of the main distinguishing factors of team logics from hyperlogics; namely the ability to refer directly to unbounded number of traces. Each hyperlogic-formula has a fixed number of trace quantifiers that delineate the traces involved in the evaluation of the formula. Another distinguishing feature of team logics lies in their ability to enrich the logical language with novel atomic formulae for stating properties of teams. The most prominent of these are the *dependence atom*  $\text{dep}(\bar{x}, \bar{y})$  (stating that the values of the variables  $\bar{x}$  functionally determine the values of  $\bar{y}$ ) and *inclusion atom*  $\bar{x} \subseteq \bar{y}$  (expressing the inclusion dependency that all the values occurring for  $\bar{x}$  must also occur as a value for  $\bar{y}$ ).

As an example, let  $o_1, \dots, o_n$  be public observables and assume that  $c$  reveals confidential information. The atom  $(o_1, \dots, o_n, c) \subseteq (o_1, \dots, o_n, \neg c)$  expresses a form of non-inference by stating that an observer cannot infer the value of the confidential bit from the outputs.

While HyperLTL and other hyperlogics have been studied extensively, many of the basic properties of TeamLTL are still not well understood. Krebs et al. [15] showed that synchronous TeamLTL and HyperLTL are incomparable in expressivity and that the asynchronous variant collapses to LTL. Not much was known about the complexity aspects of TeamLTL until Lück [18] showed that the complexity of satisfiability and model checking of synchronous TeamLTL with Boolean negation  $\sim$  is equivalent to the decision problem of third-order arithmetic. Subsequently, Virtema et al. [29] embarked for a more fine-grained analysis of the complexity of synchronous TeamLTL and discovered a decidable syntactic fragment (the so-called *left-flat fragment*) and established that already a very weak access to the Boolean negation suffices for undecidability. They also showed that synchronous TeamLTL and its extensions can be translated to  $\text{HyperQPTL}^+$ , which is an extension of HyperLTL by (non-uniform) quantification of propositions. Kontinen and Sandström [12] defined translations between extensions of TeamLTL and the three-variable fragment of first-order team logic to utilize the better understanding of first-order team semantics. They also showed that any logic effectively residing between synchronous TeamLTL extended with the Boolean negation and second-order logic inherits the complexity properties of the extension of TeamLTL with the Boolean negation. Finally, Gutsfeld et al. [7] reimagined the setting of temporal team semantics to be able to model richer forms of (a)synchronicity by developing the notion of time-evaluation functions. In addition to reimagining the framework, they discovered

decidable logics which however relied on restraining time-evaluation functions to be either *k-context-bounded* or *k-synchronous*. It is worth noting that recently asynchronous hyperlogics have been considered also in several other articles (see, e.g., [8, 1]).

Almost all complexity theoretic results previously obtained for TeamLTL have been negative, and the few positive results have required drastic restrictions in syntax or semantics. In this article we take a fresh look at expressive extensions of asynchronous TeamLTL. Recent works on synchronous TeamLTL have revealed that quite modest extensions of synchronous TeamLTL are undecidable. Thus, our study of asynchronous TeamLTL partly stems from our desire to discover decidable but expressive logics for hyperproperties. Until now, all the papers on temporal team semantics have explicitly or implicitly adopted a semantics based on multisets of traces. In the team semantics literature, this often carries the name *strict semantics*, in contrast to *lax semantics* which is de-facto set-based semantics. In database theory, it is ubiquitous that tasks that are computationally easy under set based semantics become untractable in the multiset case. In the team semantics setting this can be already seen in the model checking problem of propositional inclusion logic  $PL(\subseteq)$  which is P-complete under lax semantics, but NP-complete under strict semantics [10]. Our new set-based framework offers a setting that drops the accuracy that accompanies adoption of multiset semantics in favour of better computational properties. Consider the following formula expressing a form of strong non-inference in parallel computation:  $G((o_1, \dots, o_n, c) \subseteq (o_1, \dots, o_n, \neg c))$ , where  $o_1, \dots, o_n$  are observable outputs and  $c$  is confidential. In the synchronous setting, the formula expresses that during a synchronous computation, at any given time, an observer cannot infer the value of the secret  $c$  from the outputs. In the asynchronous setting, the formula states a stronger property that the above property holds for all computations (not only synchronous). In the multiset setting the number of parallel computation nodes is fixed, while in the new lax semantics, we drop that restriction, and consider an undefined number of computation nodes. The condition is stronger in lax semantics; and intuitively easier to falsify, which makes model checking in practice easier.

**Our contribution.** We introduce and develop a set-based semantics for asynchronous TeamLTL, which we name *lax semantics* and write  $\text{TeamLTL}^l$ . We consider two canonical logics in this setting: the extensions of  $\text{TeamLTL}^l$  by the Boolean disjunction  $\text{TeamLTL}^l(\vee)$  and by the Boolean negation  $\text{TeamLTL}^l(\sim)$ . By developing the basic theory of lax asynchronous TeamLTL, we discover some fascinating connections between the strict and lax semantics. We discover that both of the logics enjoy normal forms that can be utilised to obtain expressivity and complexity results. Tables 1 and 2 summarise our results. For comparison, Table 3 summarises the known results on complexity of synchronous TeamLTL.

## 2 Preliminaries

Fix a set AP of *atomic propositions*. The set of formulae of LTL (over AP) is generated by the grammar:  $\varphi ::= p \mid \neg p \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc \varphi \mid G \varphi \mid \varphi \text{U} \psi$ , where  $p \in \text{AP}$ . We adopt the convention that formulae are given in negation normal form, i.e.,  $\neg$  is allowed only in front of atomic propositions. Note that this is an expressively complete set of LTL-formulae. The logical constants  $\top, \perp$  and the operators  $F$  and  $W$  can be defined in the usual way:  $\perp := p \wedge \neg p$ ,  $\top := p \vee \neg p$ ,  $F \varphi := \top \text{U} \varphi$ , and  $\varphi W \psi := (\varphi \text{U} \psi) \vee G \varphi$ . Note also the equivalences  $\neg \bigcirc \varphi \equiv \bigcirc \neg \varphi$ ,  $\neg F \varphi \equiv G \neg \varphi$ , and  $\neg(\varphi \text{U} \psi) \equiv (\neg \varphi W (\neg \psi \wedge \neg \varphi))$ .

A *trace*  $t$  over AP is an infinite sequence from  $(2^{\text{AP}})^\omega$ . For a natural number  $i \in \mathbb{N}$ , we denote by  $t[i]$  the  $(i + 1)$ th letter of  $t$  and by  $t[i, \infty]$  the postfix  $(t[j])_{j \geq i}$  of  $t$ . Semantics of LTL is defined in the usual manner (see e.g., [21]). For example,  $t \models p$  iff  $p \in t[0]$  and  $t \models \bigcirc \varphi$  iff  $t[1, \infty] \models \varphi$ . The *truth value* of a formula  $\varphi$  on a trace  $t$  is denoted by  $\llbracket \varphi \rrbracket_t \in \{0, 1\}$ .

■ **Table 1** Expressivity hierarchy of the asynchronous logics considered in the paper. Logics with lax or strict semantics are here referred with the superscripts  $l$  and  $s$ , respectively. For the definitions of left flatness, quasi flatness, and left downward closure, we refer to Definitions 7 and 13. †: This follows since only  $\text{TeamLTL}^l(\otimes)$  is downward closed (cf. Theorem 8 and Definition 13). Theorem 8 implies that for  $\text{TeamLTL}(\sim)$ -formulae in quasi-flat form the strict and lax semantics coincide.

$\text{TeamLTL}^{s/l}$		left-flat- $\text{TeamLTL}^s(\otimes)$	$\stackrel{\text{Cor. 12}}{<}$	$\text{TeamLTL}^s(\otimes)$
$\wedge$ Ex. 6		$\equiv$ Thm. 8		
$\text{TeamLTL}^l(\otimes)$	$\stackrel{\text{Thm. 10}}{\equiv}$	left-flat- $\text{TeamLTL}^l(\otimes)$	$\stackrel{\dagger}{<}$	quasi-flat- $\text{TeamLTL}^{s/l}(\sim)$
				$\equiv$ Thm. 14
				left-dc- $\text{TeamLTL}^l(\sim)$

■ **Table 2** Complexity results of this paper. All results are completeness results if not otherwise specified.  $\text{PL}(\sim)$  refers to the propositional fragment of  $\text{TeamLTL}(\sim)$  which embeds also to left-dc- $\text{TeamLTL}^l(\sim)$ . †: All PSPACE-completeness results for satisfiability in strict semantics and  $\text{TeamLTL}^l$  follow directly from classical LTL by downward closure and singleton equivalence similar to [15, Proposition 5.4].  $\text{ATIME-ALT}(\text{exp, poly})$  refers to alternating exponential time with polynomially many alternations while  $\text{TOWER}(\text{poly})$  refers to problems that can be decided by a deterministic TM in time bounded by an exponential tower of 2's of polynomial height.

Logic (asynchronous semantics)	Complexity of		References
	model checking	satisfiability	
LTL	PSPACE	PSPACE	[25]
$\text{PL}(\sim)$	$\text{ATIME-ALT}(\text{exp, poly})$	$\text{ATIME-ALT}(\text{exp, poly})$	[9]
$\text{TeamLTL}^{l/s}$	PSPACE	PSPACE	[15], Theorem 5
left-flat- $\text{TeamLTL}^{s/l}(\otimes)$	PSPACE	PSPACE	Theorem 17
$\text{TeamLTL}^l(\otimes)$	PSPACE	PSPACE	Theorem 17
$\text{TeamLTL}^s(\otimes)$	???	PSPACE	†
$\text{TeamLTL}^s(\text{dep})$	NEXPTIME-hard	PSPACE	[15]
left-dc- $\text{TeamLTL}^l(\sim)$	in $\text{TOWER}(\text{poly})$	in $\text{TOWER}(\text{poly})$	Theorem 17

■ **Table 3** Complexity results for synchronous strict semantics. All results are completeness results if not otherwise specified. †: All PSPACE-completeness results for satisfiability follow directly from classical LTL by downward closure and singleton equivalence similar to [15, Proposition 5.4].

Logic (sync. strict semantics)	Complexity of		References
	model checking	satisfiability	
$\text{TeamLTL}$	PSPACE	PSPACE	[15]
left-flat- $\text{TeamLTL}(\otimes)$	in EXSPACE	PSPACE	[29]
$\text{TeamLTL}(\text{dep})$	NEXPTIME-hard	PSPACE	[15]
$\text{TeamLTL}(\otimes)$	???	PSPACE	†
$\text{TeamLTL}(\otimes, \subseteq)$	$\Sigma_1^0$ -hard	$\Sigma_1^0$ -hard	[29]
$\text{TeamLTL}(\sim)$	third-order arithmetic	third-order arithmetic	[18]

Next we present the so-called asynchronous team semantics for LTL introduced in [15]. In [15], the release operator was defined slightly erroneously; we fix the issue here by taking  $G$  as primitive and defining  $R$  using  $G$  and  $U$ . Informally, a multiset of traces  $T$  is a collection of traces with possible repetitions. Formally, we represent  $T$  as a set of pairs  $(i, t)$ , where  $i$  is an *index* (from some suitable large set) and  $t$  is a trace. We stipulate that the elements of a multiset have distinct indices. From now on, we will always omit the index and write  $t$  instead of  $(i, t)$ . For multisets  $T$  and  $S$ ,  $T \uplus S$  denotes the disjoint union of  $T$  and  $S$  (obtained by stipulating that traces in  $S$  and  $T$  have disjoint sets of indices). Note that all the functions  $f$  with domain  $T$  are actually of the form  $f((i, t))$  and may map different copies of the trace  $t$  differently. A *team* (*multiteam*, resp.) is a set (multiset, resp.) of traces. If  $f: T \rightarrow \mathbb{N}$  is a function, we define the updated team  $T[f, \infty] := \{t[f(t), \infty] \mid t \in T\}$ , where  $f$  determines for each trace a point in time it updates to. For functions  $f$  and  $f'$  as above, we write  $f' < f$ , if  $f'(t) < f(t)$  for all  $t \in T$ . The underlying team support( $T$ ) :=  $\{t \mid (i, t) \in T\}$  of a multiteam  $T$  is called the *support* of  $T$ .

► **Definition 1** (Team Semantics for LTL). *Let  $T$  be a multiteam, and  $\varphi$  and  $\psi$  LTL-formulae. The asynchronous team semantics of TeamLTL is defined as follows.*

$$\begin{aligned}
T \models l & \Leftrightarrow t \models l \text{ for all } t \in T, \text{ where } l \in \{p, \neg p \mid p \in \text{AP}\} \text{ is a literal} \\
T \models \varphi \wedge \psi & \Leftrightarrow T \models \varphi \text{ and } T \models \psi \\
T \models \varphi \vee \psi & \Leftrightarrow \exists T_1, T_2 \text{ s.t. } T_1 \uplus T_2 = T \text{ and } T_1 \models \varphi \text{ and } T_2 \models \psi \\
T \models \bigcirc \varphi & \Leftrightarrow T[1, \infty] \models \varphi, \text{ where } 1 \text{ is the constant function } t \mapsto 1 \\
T \models G\varphi & \Leftrightarrow \forall f: T \rightarrow \mathbb{N} \quad T[f, \infty] \models \varphi \\
T \models \varphi \text{ U } \psi & \Leftrightarrow \exists f: T \rightarrow \mathbb{N} \quad T[f, \infty] \models \psi \text{ and } \forall f' < f: T'[f', \infty] \models \varphi, \\
& \text{ where } T' := \{t \in T \mid f(t) \neq 0\}
\end{aligned}$$

The synchronous variant of the semantics is obtained by allowing  $f$  to range only over constant functions. We take the asynchronous semantics as the standard semantics and write TeamLTL for asynchronous TeamLTL.

We also consider the *Boolean disjunction*  $\otimes$  and *Boolean negation*  $\sim$  interpreted as usual:  $T \models \varphi \otimes \psi$  iff  $(T \models \varphi \text{ or } T \models \psi)$ , and  $T \models \sim \varphi$  iff  $T \not\models \varphi$ .

Next we define some important semantic properties of formulae studied in the literature. A logic has one of the properties if every formula of the logic has the property. It is easy to check that TeamLTL has all the properties listed [15] whereas its extension with the Boolean disjunction has all but flatness and the extension with Boolean negation has none.

**(Downward closure)** If  $T \models \varphi$  and  $S \subseteq T$ , then  $S \models \varphi$ .

**(Empty team property)**  $\emptyset \models \varphi$ .

**(Flatness)**  $T \models \varphi$  iff  $\{t\} \models \varphi$  for all  $t \in T$ .

**(Singleton equivalence)**  $\{t\} \models \varphi$  iff  $t \models \varphi$ .

We will now justify our choice of semantics. The semantic rules for literals, conjunction, and disjunction are the standard ones in team semantics, and which have been motivated numerous times in the literature [26]. Two important properties for the logic to have, for it to be a conservative extension of LTL, are flatness and singleton equivalence. These properties also motivated the original definition of asynchronous TeamLTL [15]. The given semantics for  $\bigcirc$  is the only possible one that satisfies flatness. The same is true for  $F$  (i.e.,  $\top \text{ U } \varphi$ ) and  $G$ ; moreover the semantics clearly capture the intuitive meanings of asynchronously in the future and asynchronously globally, respectively. The given semantics for  $U$  preserves flatness

and singleton equivalence, and adequately captures the intuitive meaning of asynchronous until. The framework of asynchronous TeamLTL then allows us to define different variants of the familiar temporal operators. E.g.,  $\varphi W_1 \psi := G \varphi \vee \varphi U \psi$  and  $\varphi W_2 \psi := G \varphi \otimes \varphi U \psi$  define different variants of *weak until*; the first of which is flat, while the second is not.

$$\begin{aligned} T \models \varphi W_1 \psi &\Leftrightarrow \exists T_1, T_2 \text{ s.t. } T_1 \uplus T_2 = T, T_1 \models G \varphi \text{ and } T_2 \models \varphi U \psi \\ T \models \varphi W_2 \psi &\Leftrightarrow T \models G \varphi \text{ or } T \models \varphi U \psi \end{aligned}$$

Similarly  $\varphi R_1 \psi := \psi U((\psi \wedge \varphi) \vee G \psi)$  and  $\varphi R_2 \psi := \psi U((\psi \wedge \varphi) \otimes G \psi)$  give rise to different variants of *release*. Moreover, with  $\sim$  one can define additional dual operators.

A defining feature of team semantics is the ability to enrich logics with novel atomic statements describing properties of teams in a modular fashion. For example, *dependence atoms*  $\text{dep}(\varphi_1, \dots, \varphi_n, \psi)$  and *inclusion atoms*  $\varphi_1, \dots, \varphi_n \subseteq \psi_1, \dots, \psi_n$ , with  $\varphi_1, \dots, \varphi_n, \psi, \psi_1, \dots, \psi_n$  being LTL-formulae, have been studied extensively in first-order and modal team semantics. The dependence atom states that the truth value of  $\psi$  is functionally determined by that of  $\varphi_1, \dots, \varphi_n$  whereas the inclusion atom states that each value combination of  $\varphi_1, \dots, \varphi_n$  must also occur as a value combination for  $\psi_1, \dots, \psi_n$ . Formally:

$$T \models \text{dep}(\varphi_1, \dots, \varphi_n, \psi) \text{ iff } \forall t, t' \in T : \left( \bigwedge_{1 \leq j \leq n} \llbracket \varphi_j \rrbracket_t = \llbracket \varphi_j \rrbracket_{t'} \right) \Rightarrow \llbracket \psi \rrbracket_t = \llbracket \psi \rrbracket_{t'}$$

$$T \models \varphi_1, \dots, \varphi_n \subseteq \psi_1, \dots, \psi_n \text{ iff } \forall t \in T \exists t' \in T : \bigwedge_{1 \leq j \leq n} \llbracket \varphi_j \rrbracket_t = \llbracket \psi_j \rrbracket_{t'}$$

Consider the following exemplary formula:  $G \text{dep}(i_1, i_2, o) \vee G \text{dep}(i_2, i_3, o)$ . The formula states that the executions of the system can be decomposed into two parts; in the first part, the output  $o$  is determined by the inputs  $i_1$  and  $i_2$ , and in the second part,  $o$  is determined by the inputs  $i_2$  and  $i_3$ .

If  $\mathcal{A}$  is a collection of atoms and connectives,  $\text{TeamLTL}(\mathcal{A})$  denotes the extension of TeamLTL with the atoms and connectives in  $\mathcal{A}$ . It is straightforward to see (in analogy to the modal team semantics setting [11]) that any dependency such as the ones above is determined by a finite set of  $n$ -ary Boolean relations. Let  $B$  be a set of  $n$ -ary Boolean relations. We define the property  $[\varphi_1, \dots, \varphi_n]_B$  for an  $n$ -tuple  $(\varphi_1, \dots, \varphi_n)$  of LTL-formulae:

$$T \models [\varphi_1, \dots, \varphi_n]_B \text{ iff } \{(\llbracket \varphi_1 \rrbracket_t, \dots, \llbracket \varphi_n \rrbracket_t) \mid t \in T\} \in B.$$

Expressions of the form  $[\varphi_1, \dots, \varphi_n]_B$  are *generalised atoms*. It was shown in [29] that, in the synchronous setting,  $\text{TeamLTL}(\sim)$  is expressively complete with respect to all generalised atoms, whereas the extension of  $\text{TeamLTL}(\otimes)$  with the so-called *flattening operator* can express any downwards closed generalised atoms. These results readily extend to the asynchronous setting. Moreover the flattening operator renders itself unnecessary due to flatness of asynchronous TeamLTL. The results imply, e.g, that the (downwards closed) dependence atoms can be expressed in both of the logics  $\text{TeamLTL}(\sim)$  and  $\text{TeamLTL}(\otimes)$ , and inclusion atoms in turn are expressible in  $\text{TeamLTL}(\sim)$ . The proof of the following theorem is essentially the same as the proof of [28, Proposition 17]. Below  $L \equiv L'$  denotes the equiexpressivity of the logics  $L$  and  $L'$ .

► **Theorem 2.** *Let  $\mathcal{A}, \mathcal{D}$  be the sets of all generalised atoms, and all downward closed generalised atoms. Then  $\text{TeamLTL}(\mathcal{D}, \otimes) \equiv \text{TeamLTL}(\otimes)$  and  $\text{TeamLTL}(\mathcal{A}, \sim) \equiv \text{TeamLTL}(\sim)$ .*

### 3 Set-based semantics for TeamLTL

Next we define a relaxed version of the asynchronous semantics. We call it *lax* semantics as it corresponds to the so-called lax semantics of first-order team semantics (see e.g., [6]). From now on we refer to the semantics of Definition 1 as strict semantics. The possibility of considering lax semantics for TeamLTL was suggested by Lück already in [19] but the full definition was not given. Intuitively, lax semantics can always be obtained from a strict one by checking what strict semantics would yield if multiteams were enriched with unbounded many copies of each of its traces. One of the defining features of lax semantics is that it is unable to distinguish multiplicities, which is formalised by Proposition 4 below.

We need some notation for the new definition. We write  $\mathcal{P}(\mathbb{N})^+$  to denote  $\mathcal{P}(\mathbb{N}) \setminus \{\emptyset\}$ . For a team  $T$  and function  $f: T \rightarrow \mathcal{P}(\mathbb{N})^+$ , we set  $T[f, \infty] := \{t[s, \infty] \mid t \in T, s \in f(t)\}$ . For  $T' \subseteq T$ ,  $f: T \rightarrow \mathcal{P}(\mathbb{N})^+$ , and  $f': T' \rightarrow \mathcal{P}(\mathbb{N})^+$ , we define that  $f' < f$  if and only if

$$\forall t \in T': \min(f'(t)) \leq \min(f(t)) \text{ and, if } \max(f(t)) \text{ exists, } \max(f'(t)) < \max(f(t)).$$

► **Definition 3** (TeamLTL<sup>l</sup>). *Let  $T$  be a team, and  $\varphi$  and  $\psi$  TeamLTL-formulae. The lax semantics is defined as follows. We only list the cases that differ from the strict semantics.*

$$\begin{aligned} T \models^l \varphi \vee \psi &\Leftrightarrow \exists T_1, T_2 \text{ s.t. } T_1 \cup T_2 = T \text{ and } T_1 \models \varphi \text{ and } T_2 \models \psi \\ T \models^l \mathbf{G} \varphi &\Leftrightarrow \forall f: T \rightarrow \mathcal{P}(\mathbb{N})^+ \text{ it holds that } T[f, \infty] \models^l \varphi \\ T \models^l \varphi \cup \psi &\Leftrightarrow \exists f: T \rightarrow \mathcal{P}(\mathbb{N})^+ \text{ such that } T[f, \infty] \models^l \psi \text{ and} \\ &\quad \forall f': T' \rightarrow \mathcal{P}(\mathbb{N})^+ \text{ s.t. } f' < f, \text{ it holds that } T'[f', \infty] \models^l \varphi \text{ or } T' = \emptyset, \\ &\quad \text{where } T' := \{t \in T \mid \max(f(t)) \neq 0\} \end{aligned}$$

In the context we will be considering in this article, the subformulae  $\varphi$  in the definition of the until operator  $\mathbf{U}$  always have the empty team property and thus we disregard the possibility that the team  $T'$  is empty in our proofs, as that case follows from the empty team property.

The above set-based semantics can also be viewed in terms of multisets. In that case functions  $f$  are quantified uniformly, i.e. we restrict our consideration to functions where  $f(i, t) = f(j, t)$ . Furthermore, the semantics for disjunction is defined in a way that omits references to multiplicities. In order to relate our new logics to the old multiteam based ones, we extend the lax semantics to multiteams  $T$  by stipulating that  $T \models^l \varphi$  iff  $\text{support}(T) \models^l \varphi$ .

The following proposition shows that TeamLTL<sup>l</sup>( $\sim$ ) satisfies the so-called *locality* property, see full version of this article [13] for the proof. For a trace  $t$  over  $\text{AP}'$  and  $\text{AP} \subseteq \text{AP}'$ , the *reduction* of  $t$  to  $\text{AP}$ ,  $t_{\uparrow \text{AP}}$ , is a sequence from  $(2^{\text{AP}})^\omega$  such that  $p \in t[i]$  if and only if  $p \in t_{\uparrow \text{AP}}[i]$ , for all  $p \in \text{AP}$  and  $i \in \mathbb{N}$ . For a team  $T$  over  $\text{AP}'$  we define the *reduction* of  $T$  to  $\text{AP}$  by  $T_{\uparrow \text{AP}} = \{t_{\uparrow \text{AP}} \mid t \in T\}$ .

► **Proposition 4.** *Let  $T$  be a team and  $\varphi$  a TeamLTL<sup>l</sup>( $\sim$ )-formula with variables in  $\text{AP}$ . Now  $T \models^l \varphi$  iff  $T_{\uparrow \text{AP}} \models^l \varphi$ .*

The next theorem displays that lax semantics enjoys the same fundamental properties as its strict counterpart. The proof via a straightforward induction, see full version of this article [13] for details.

► **Theorem 5.** *TeamLTL<sup>l</sup> satisfies downward closure, empty team property, singleton equivalence, and flatness.*

The following example establishes that the new lax semantics differs from the strict semantics, and that in the old semantics multiplicities matter. Moreover, we obtain TeamLTL<sup>l</sup> < TeamLTL<sup>l</sup>( $\otimes$ ) by showing that the latter is not flat.

► **Example 6.** Let  $\varphi$  be the formula  $G(p \otimes q)$ ,  $T_1 := \{t\}$  and  $T_2 := \{(1, t), (2, t)\}$ , where  $t := \{p\}\{q\}^\omega$ . It is easy to check that  $T_1 \models \varphi$  but  $T_1 \not\models^l \varphi$  (which is witnessed by  $T[f, \infty] \not\models^l p \otimes q$  for  $f(t) := \{0, 1\}$ ). Likewise  $T_2 \not\models \varphi$ . Moreover  $\{s_i\} \models^l \varphi$ , for  $i \in \{1, 2\}$ , but  $\{s_1, s_2\} \not\models^l \varphi$ , where  $s_1 := \{p\}^\omega$  and  $s_2 := \{q\}^\omega$ .

We will also consider the following fragments of  $\text{TeamLTL}(\otimes)$  and  $\text{TeamLTL}(\sim)$ .

► **Definition 7.** A formula  $\varphi$  of  $\text{TeamLTL}(\otimes)$  is called *left-flat*, if in all of its subformulae of the form  $G\psi$  and  $\psi \cup \theta$ , the subformula  $\psi$  is an LTL-formula. A formula  $\varphi$  of  $\text{TeamLTL}(\sim, \otimes)$  is called *left-downward closed*, if in all of its subformulae of the form  $G\psi$  and  $\psi \cup \theta$ , the subformula  $\psi$  is an  $\text{TeamLTL}(\otimes)$ -formula.

We will later show that the above syntactic restriction for flatness could be replaced by a semantic restriction (see Corollary 11). The proof of the following theorem is in the full version of this article [13].

► **Theorem 8.** For all  $\varphi \in \text{TeamLTL}^l(\otimes)$  the following two claims hold:

1.  $\varphi$  is downward closed and has the empty team property, and
2. if  $\varphi$  is left-flat, then  $T \models \varphi$  iff  $\text{support}(T) \models^l \varphi$  for all multiteams  $T$ .

The restriction to left-flat formulae in case (2) above is necessary by Example 6.

## 4 Normal Forms for TeamLTL with Boolean Disjunction and Negation

In this section we develop normal forms for our logics, which we then utilise to obtain strong expressivity and complexity results.

► **Definition 9.** A formula  $\varphi$  is in  $\otimes$ -disjunctive normal form if it is of the form  $\bigvee_{i \in I} \alpha_i$ , where  $\alpha_i$  are LTL-formulae.

Every formula of  $\text{TeamLTL}^l(\otimes)$  can be transformed into an equivalent  $\otimes$ -disjunctive normal form. This result is similar to the one proved in [27] for team-based modal logic  $\text{ML}(\otimes)$ . In the following  $|\varphi|$  denotes the length of the formula  $\varphi$ .

► **Theorem 10.** Every  $\varphi \in \text{TeamLTL}^l(\otimes)$  is logically equivalent to a formula  $\varphi^* = \bigvee_{i \in I} \alpha_i$  in  $\otimes$ -disjunctive normal form, where  $|\alpha_i| \leq |\varphi|$  and  $|I| = 2^k$ , where  $k$  is the number of  $\otimes$  in  $\varphi$ .

**Proof.** The proof proceeds by induction on the structure of formulae. Note that atomic formulae are already in the normal form and that the case for  $\otimes$  is trivial. The remaining cases are defined as follows:

$$\begin{aligned} (\psi \wedge \theta)^* &:= \bigvee_{i \in I, j \in J} (\alpha_i^\psi \wedge \alpha_j^\theta) & (\psi \vee \theta)^* &:= \bigvee_{i \in I, j \in J} (\alpha_i^\psi \vee \alpha_j^\theta) \\ (\bigcirc \psi)^* &:= \bigvee_{i \in I} \bigcirc \alpha_i^\psi & (G\psi)^* &:= \bigvee_{i \in I} G \alpha_i^\psi \\ (\psi \cup \theta)^* &:= \bigvee_{i \in I, j \in J} (\alpha_i^\psi \cup \alpha_j^\theta). \end{aligned}$$

where  $\alpha_i^\psi$  and  $\alpha_j^\theta$  are the flat formulae in the disjunctive normal forms of  $\psi$  and  $\theta$  respectively, and  $I$  and  $J$  are the respective index sets.

Suppose  $\varphi = \psi \wedge \theta$  and that  $\psi \equiv \bigvee_{i \in I} \alpha_i^\psi$  and  $\theta \equiv \bigvee_{j \in J} \alpha_j^\theta$  (induction hypothesis). Now  $T \models^l \varphi$  if and only if  $T \models^l \psi$  and  $T \models^l \theta$ . The latter holds, if and only if  $T \models^l \alpha_k^\psi$  and  $T \models^l \alpha_{k'}^\theta$ , for some  $k$  and  $k'$ . This can be equivalently expressed as  $T \models^l \bigvee_{i, j} (\alpha_i^\psi \wedge \alpha_j^\theta)$ , i.e.  $T \models^l \varphi^*$ .

Suppose  $\varphi = \psi \vee \theta$  and that  $\psi \equiv \bigvee_{i \in I} \alpha_i^\psi$  and  $\theta \equiv \bigvee_{j \in J} \alpha_j^\theta$ . By definition  $T \models^l \varphi$  if and only if there exists  $T' \cup T'' = T$  such that  $T' \models^l \psi$  and  $T'' \models^l \theta$ . By the induction hypothesis the latter is equivalent with  $T' \models^l \bigvee_{i \in I} \alpha_i^\psi$  and  $T'' \models^l \bigvee_{j \in J} \alpha_j^\theta$ . By definition this holds if and only if there are  $k'$  and  $k''$  such that  $T' \models^l \alpha_{k'}^\psi$  and  $T'' \models^l \alpha_{k''}^\theta$ , which is equivalent with  $T \models^l \alpha_{k'}^\psi \vee \alpha_{k''}^\theta$ , for some  $k'$  and  $k''$ , by definition. Equivalently then  $T \models^l \bigvee_{i \in I, j \in J} (\alpha_i^\psi \vee \alpha_j^\theta)$ .

Suppose  $\varphi = \bigcirc \psi$  and that  $\psi \equiv \bigvee_{i \in I} \alpha_i^\psi$ . By definition  $T \models^l \varphi$  is equivalent with  $T[1, \infty] \models^l \psi$ . By the induction hypothesis the latter holds if and only if  $T[1, \infty] \models^l \bigvee_{i \in I} \alpha_i^\psi$ , which by definition is equivalent with  $T[1, \infty] \models^l \alpha_k^\psi$  for some  $k \in I$ . The latter holds if and only if  $T \models^l \bigcirc \alpha_k^\psi$  for some  $k \in I$ , which is equivalent with  $T \models^l \bigvee_{i \in I} \bigcirc \alpha_i^\psi$ .

Suppose  $\varphi = \mathbf{G} \psi$  and that  $\psi \equiv \bigvee_{i \in I} \alpha_i^\psi$ . Suppose that  $T \models^l \varphi$ . By definition for all functions  $f: T \rightarrow \mathcal{P}(\mathbb{N})^+$  it holds that  $T[f, \infty] \models^l \psi$ . By the induction hypothesis  $T[f, \infty] \models^l \bigvee_{i \in I} \alpha_i^\psi$  for all  $f$ . Especially this holds for the total function defined for every  $t \in T$  by  $f_{max}(t) := \mathbb{N}$ . Thus  $T[f_{max}, \infty] \models^l \alpha_k^\psi$  for some  $k$ . By downward closure it holds that  $T[f', \infty] \models^l \alpha_k^\psi$  for all  $f': T \rightarrow \mathcal{P}(\mathbb{N})^+$ . Hence  $T \models^l \mathbf{G} \alpha_k^\psi$ , and thus  $T \models^l \bigvee_{i \in I} \mathbf{G} \alpha_i^\psi$ . The other direction is analogous.

Suppose  $\varphi = \psi \mathbf{U} \theta$  and that  $\psi \equiv \bigvee_{i \in I} \alpha_i^\psi$  and  $\theta \equiv \bigvee_{j \in J} \alpha_j^\theta$ . Suppose  $T \models^l \varphi$ . By definition there exists a function  $f: T \rightarrow \mathcal{P}(\mathbb{N})^+$  such that  $T[f, \infty] \models^l \theta$  and for all functions  $f': T' \rightarrow \mathcal{P}(\mathbb{N})^+$  such that  $f' < f$ ,  $T'[f', \infty] \models^l \psi$ , where  $T' := \{t \in T \mid f(t) \neq 0\}$ . Hence by the induction hypothesis  $T[f, \infty] \models^l \bigvee_{j \in J} \alpha_j^\theta$ , which is equivalent with  $T[f, \infty] \models^l \alpha_k^\theta$  for some  $k \in J$ , and, for the function  $f_{max}(t) := \{n \in \mathbb{N} \mid n < m, \text{ for some } m \in f(t)\}$  (which is well-defined, as  $f(t)$  is non-empty for  $t \in T'$ ), it holds that  $T[f_{max}, \infty] \models^l \bigvee_{i \in I} \alpha_i^\psi$ , which in turn is equivalent with  $T[f_{max}, \infty] \models^l \alpha_{k'}^\psi$  for some  $k' \in I$ . By downward closure the latter holds for all intermediary functions, and thus  $T \models^l \alpha_{k'}^\psi \mathbf{U} \alpha_k^\theta$  and finally  $T \models^l \bigvee_{i \in I, j \in J} (\alpha_i^\psi \mathbf{U} \alpha_j^\theta)$  as wanted. The converse is analogous.

For showing the size estimates stated in the theorem, it suffices to note that our translation to  $\bigcirc$ -disjunctive normal form can be equivalently stated:  $\varphi \equiv \bigvee_{i \in I} \alpha_i^\psi = \bigvee_{f \in F} \varphi^f$ , where  $F$  is the set of all selection functions  $f$  that select, separately for each occurrence, either the left disjunct  $\psi$  or the right disjunct  $\theta$  of each subformula of the form  $\psi \bigcirc \theta$  of  $\varphi$ , and  $\varphi^f$  denotes the formula obtained from  $\varphi$  by substituting each occurrence of a subformula of type  $(\psi \bigcirc \theta)$  by  $f(\psi \bigcirc \theta)$ . The size estimates follow immediately from this observation.  $\blacktriangleleft$

Proofs for the following two corollaries can be found in the full version of this article [13].

► **Corollary 11.** *For every flat TeamLTL<sup>l</sup>( $\bigcirc$ )-formula there exists an equivalent TeamLTL<sup>l</sup>-formula.*

► **Corollary 12.**  $\text{TeamLTL}^l(\bigcirc) < \text{TeamLTL}(\bigcirc)$ .

A normal form, similar to the one in Theorem 10, can also be obtained for TeamLTL( $\sim$ ). However, since the extension is not downward closed, it only holds for a specific fragment of the logic. The following normal form has been introduced and used in [17, 16] to analyse the complexity of modal team logic and FO<sup>2</sup> in the team semantics context. Below  $\varphi^d$  denotes a formula obtained by transforming  $\neg \varphi$  into negation normal form in the standard way in LTL.

► **Definition 13.** *A formula  $\varphi$  is quasi-flat if  $\varphi$  is of the form:  $\bigvee_{i \in I} (\alpha_i \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j})$ , where  $\alpha_i$  and  $\beta_{i,j}$  are LTL-formulae, and  $\exists \beta_{i,j}$  is an abbreviation for the formula  $\sim \beta_{i,j}^d$ .*

Note that, for LTL-formulae  $\alpha$  and  $\beta$ , we have  $T \models^l \alpha$  if and only if  $t \models \alpha$ , for all  $t \in T$ . Moreover  $T \models^l \exists \beta$ , if and only if there exists some trace  $t \in T$  such that  $t \models \beta$ .

► **Theorem 14.** *Every left-downward closed formula  $\varphi \in \text{TeamLTL}^l(\sim, \otimes)$  is logically equivalent to a quasi-flat formula  $\varphi^*$ .*

**Proof.** Proof by induction over the structure of  $\varphi$ . Atoms are flat, and hence are in the normal form. The translations and the proofs of correctness for the cases of conjunction, disjunction, and Boolean negation are analogous to the simpler modal framework of [17, 16].

Suppose  $\varphi = \psi \wedge \theta$  and assume that  $\psi$  is equivalent to  $\bigotimes_{i \in I} (\alpha_i^\psi \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j}^\psi)$  and  $\theta$  to  $\bigotimes_{i \in I'} (\alpha_i^\theta \wedge \bigwedge_{j \in J'_i} \exists \beta_{i,j}^\theta)$ . By the distributive laws of conjunction and disjunction,  $\varphi$  is clearly equivalent to

$$\bigotimes_{i \in I, k \in I'} (\alpha_i^\psi \wedge \alpha_k^\theta \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j}^\psi \wedge \bigwedge_{j \in J'_k} \exists \beta_{k,j}^\theta).$$

Suppose  $\varphi = \psi \vee \theta$ . By the induction hypothesis and an argument analogous to the disjunction case of the proof of Theorem 10,  $\varphi$  is equivalent to

$$\bigotimes_{i \in I, k \in I'} ((\alpha_i^\psi \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j}^\psi) \vee (\alpha_k^\theta \wedge \bigwedge_{j \in J'_k} \exists \beta_{k,j}^\theta)). \quad (1)$$

The above formula expresses that  $T$  can be split into two parts:  $T_1$  in which each trace satisfies  $\alpha_i$  and the subformulae  $\beta_{i,j}$  are satisfied by some traces, and  $T_2$  in which each trace satisfies  $\alpha_k$  and the subformulae  $\beta_{k,j}$  are satisfied by some traces. But this is equivalent to saying that  $T$  can be split into two parts:  $T_1$  in which each trace satisfies  $\alpha_i$ , and  $T_2$  in which each trace satisfies  $\alpha_k$ ; and the subformulae  $\alpha_i \wedge \beta_{i,j}$  and  $\alpha_k \wedge \beta_{k,j}$  are satisfied by some traces in  $T$ , and thus the formula (1) is equivalent with

$$\bigotimes_{i \in I, k \in I'} ((\alpha_i^\psi \vee \alpha_k^\theta) \wedge \bigwedge_{j \in J_i} \exists (\alpha_i^\psi \wedge \beta_{i,j}^\psi) \wedge \bigwedge_{j \in J'_k} \exists (\alpha_k^\theta \wedge \beta_{k,j}^\theta))$$

that is in the normal form.

Suppose  $\varphi = \sim \psi$  and assume that  $\psi$  is equivalent to  $\bigotimes_{i \in I} (\alpha_i \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j})$ . Now  $\varphi$  is clearly equivalent to  $\bigwedge_{i \in I} (\exists \alpha_i^d \otimes \bigotimes_{j \in J_i} \beta_{i,j}^d)$ . This formula can be expanded back to the normal form with exponential blow-up using the distributivity law of propositional logic.

Suppose  $\varphi = \bigcirc \psi$  and assume that  $\psi$  is equivalent to  $\bigotimes_{i \in I} (\alpha_i \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j})$ . It is now easy to check that  $\varphi$  is equivalent to  $\bigotimes_{i \in I} (\bigcirc \alpha_i \wedge \bigwedge_{j \in J_i} \exists \bigcirc \beta_{i,j})$ .

Suppose  $\varphi = \mathbf{G} \psi$ . Since  $\varphi$  is left-downward closed,  $\psi$  is equivalent with a formula of the form  $\bigotimes_i \alpha_i$ , which can be transformed to the normal form by Theorem 10.

Suppose  $\varphi = \psi \mathbf{U} \theta$ . By assumption  $\varphi$  is left-downward closed hence  $\psi$  is equivalent with a formula of the form  $\bigotimes_{i \in I} \alpha_i^\psi$  (by the previous theorem) and  $\theta$  is equivalent to  $\bigotimes_{k \in I'} (\alpha_k^\theta \wedge \bigwedge_{j \in J_k} \exists \beta_{k,j}^\theta)$ . Now using the fact that  $\psi$  is downward closed, it is easy to see that  $\varphi$  is logically equivalent with the formula:

$$\bigotimes_{i \in I, k \in I'} (\alpha_i^\psi \mathbf{U} (\alpha_k^\theta \wedge \bigwedge_{j \in J_k} \exists \beta_{k,j}^\theta)). \quad (2)$$

It now suffices to show that the disjuncts (for any  $i \in I, k \in I'$ ) of (2) can be equivalently expressed as:

$$(\alpha_i^\psi \mathbf{U} \alpha_k^\theta \wedge \bigwedge_{j \in J_k} \exists (\alpha_i^\psi \mathbf{U} (\alpha_k^\theta \wedge \beta_{k,j}^\theta))). \quad (3)$$

We will show the logical implication from (3) to (2). Assume

$$T \models^l (\alpha_i^\psi \mathbf{U} \alpha_k^\theta \wedge \bigwedge_{j \in J_k} \exists (\alpha_i^\psi \mathbf{U} (\alpha_k^\theta \wedge \beta_{k,j}^\theta))).$$

Let  $f$  be such that  $T[f, \infty] \models^l \alpha_k^\theta$  and that  $T[g, \infty] \models^l \alpha_i^\psi$ , for all  $g < f$ . In order to show

$$T \models^l \alpha_i^\psi \text{U}(\alpha_k^\theta \wedge \bigwedge_{j \in J_k} \exists \beta_{k,j}^\psi), \quad (4)$$

we need to make sure that traces witnessing the truth of the formulae  $\exists \beta_{k,j}^\psi$  can be found in  $T[f, \infty]$ . Here we can use the assumption that  $T \models^l \bigwedge_{j \in J_k} \exists(\alpha_i^\psi \text{U}(\alpha_k^\theta \wedge \beta_{k,j}^\psi))$  implying that for each  $j \in J_k$  there exists  $t_j \in T$  such that  $t_j \models \alpha_i^\psi \text{U}(\alpha_k^\theta \wedge \beta_{k,j}^\psi)$ . Let now  $n_j$  be such that  $t_j[n_j, \infty] \models \alpha_k^\theta \wedge \beta_{k,j}^\psi$  and that  $t_j[l, \infty] \models \alpha_i^\psi$  for all  $l < n_j$ . Now by the flatness of the formulae  $\alpha_i^\psi$ ,  $\alpha_k^\theta$ , and  $\beta_{k,j}^\psi$ , the function  $f'$  defined by

$$f'(t) := \begin{cases} f(t) \cup \{t_j[n_j, \infty]\} & \text{if } t = t_j, \text{ for some } j \in J_k \\ f(t) & \text{otherwise} \end{cases}$$

witnesses (4). The converse is proved analogously.  $\blacktriangleleft$

The following example indicates that the restriction to left-downward closed formulae is necessary for the proof to work in the above theorem. An alternate proof that does not require the restriction to left-downward closed formulae may still exist.

► **Example 15.** Let  $\varphi$  be the formula  $\text{G}(\exists p_1 \otimes \exists p_2)$  and  $T := \{t\}$ , where  $t := (\{p_1\}\{p_2\})^\omega$ . It is now easy to check that  $T \models^l \varphi$  but  $T \not\models^l \text{G} \exists p_i$  for  $i \in \{1, 2\}$ .

## 5 Computational Properties

In this section we analyse the computational properties of the logics studied in the previous section. We focus on the complexity of the model checking and satisfiability problems.

For the model checking problem one has to determine whether a team of traces generated by a given finite Kripke structure satisfies a given formula. We consider Kripke structures of the form  $K := (W, R, \eta, w_0)$ , where  $W$  is a finite set of states,  $R \subseteq W^2$  a left-total transition relation,  $\eta: W \rightarrow 2^{\text{AP}}$  a labelling function, and  $w_0 \in W$  an initial state of  $W$ . A path  $\sigma$  through  $K$  is an infinite sequence  $\sigma \in W^\omega$  such that  $\sigma[0] := w_0$  and  $(\sigma[i], \sigma[i+1]) \in R$  for every  $i \geq 0$ . The *trace of  $\sigma$*  is defined as  $t(\sigma) := \eta(\sigma[0])\eta(\sigma[1]) \cdots \in (2^{\text{AP}})^\omega$ . A Kripke structure  $K$  then *generates* the trace set  $\text{Traces}(K) := \{t(\sigma) \mid \sigma \text{ is a path through } K\}$ .

► **Definition 16.** *The model checking problem of a logic  $\mathcal{L}$  is the following decision problem: Given a formula  $\varphi \in \mathcal{L}$  and a Kripke structure  $K$  over AP, determine whether  $\text{Traces}(K) \models \varphi$ . The (countable) satisfiability problem of a logic  $\mathcal{L}$  is the following decision problem: Given a formula  $\varphi \in \mathcal{L}$ , determine whether  $T \models \varphi$  for some (countable)  $T \neq \emptyset$ .*

Below we will use the fact that the model checking and satisfiability problems of LTL are PSPACE-complete [25]. Furthermore, we use the facts that the satisfiability problem of propositional team logic  $\text{PL}(\sim)$  is ATIME-ALT(exp, poly)-complete [9], and that the complexity of modal team logic is complete for the class  $\text{TOWER}(\text{poly}) := \text{TIME}(\exp_{n \circ (1)}(1))$ , where  $\exp_0(1) := 1$  and  $\exp_{k+1}(1) := 2^{\exp_k(1)}$  [17, 16].

► **Theorem 17.**

1. *The model checking and satisfiability problems of  $\text{TeamLTL}^l(\otimes)$  are PSPACE-complete.*
2. *The model checking and satisfiability problems of the left-flat fragment of  $\text{TeamLTL}(\otimes)$  are PSPACE-complete.*

3. The model checking problem of the left-downward closed fragment of  $\text{TeamLTL}^l(\sim, \otimes)$  is PSPACE-hard and it is contained in  $\text{TOWER}(\text{poly})$ .
4. The satisfiability problem of the left-downward closed fragment of  $\text{TeamLTL}^l(\sim, \otimes)$  is ATIME-ALT(exp, poly)-hard and it is contained in  $\text{TOWER}(\text{poly})$ .

**Proof.** Let us first consider the proofs of claims 1 and 2. Note that PSPACE-hardness holds already for LTL-formulae, hence it suffices to show containment in PSPACE. Furthermore, note that 2 follows immediately from 1 and Theorem 8. Assume a formula  $\varphi \in \text{TeamLTL}^l(\otimes)$  and a Kripke structure  $K$  is given as input. By Theorem 10,  $\varphi$  is logically equivalent with a formula of the form  $\bigvee_{f \in F} \varphi^f$ , where  $f$  varies over selection functions selecting, separately for each occurrence, either the left disjunct  $\psi$  or the right disjunct  $\theta$  of each subformula of the form  $\psi \otimes \theta$  of  $\varphi$ . Now, without constructing the full formula  $\bigvee_{f \in F} \varphi^f$ , using polynomial space with respect to the size of  $\varphi$  it is possible to check whether  $\text{Traces}(K) \models \varphi_f$  for some  $f \in F$ . Hence the upper bound follows from the fact that LTL model checking is in PSPACE. The upper bound for satisfiability follows analogously.

Let us then consider the proof of claim (4). The proof of claim (3) is analogous. For the lower bound it suffices to note that propositional team logic  $\text{PL}(\sim)$  is a fragment of the left-downward closed fragment of  $\text{TeamLTL}^l(\sim, \otimes)$  and hence its satisfiability problem can be trivially reduced to the satisfiability problem of the left-downward closed fragment. Therefore ATIME-ALT(exp, poly)-hardness follows by the result of [9].

For the upper bound we first transform an input formula  $\varphi$  into an equivalent quasi-flat formula of the form  $\bigvee_{i \in I} (\alpha_i \wedge \bigwedge_{j \in J_i} \exists \beta_{i,j})$ . Analogously to [17, 16], this formula can be computed in time  $\text{TIME}(\exp_{O(|\varphi|)}(1))$ . It is now easy to see that the quasi-flat formula is satisfiable iff there exists  $i \in I$ , such that  $\text{SAT}(\alpha_i \wedge \beta_{i,j}) = 1$  for all  $j \in J_i$ . Since LTL-satisfiability checking is contained in  $\text{PSPACE} \subseteq \text{TIME}(2^{n^{O(1)}})$ , the overall complexity of the above procedure is in  $\text{TIME}(\exp_{(|\varphi|^{O(1)})}(1))$ . ◀

## 6 Conclusion

We introduced a novel set-based semantics for asynchronous TeamLTL. We showed several results on the expressive power and complexity of the extensions of  $\text{TeamLTL}^l$  by the Boolean disjunction  $\text{TeamLTL}^l(\otimes)$  and by the Boolean negation  $\text{TeamLTL}^l(\sim)$ . In particular, our results show that the complexity properties of the former logic are comparable to that of LTL and that the left-downward closed fragment of the latter has also decidable model-checking and satisfiability problems. See Table 1 on page 4 for an overview of our expressivity results and Table 2 for our complexity results. We obtained these results on  $\text{TeamLTL}^l(\otimes)$  and  $\text{TeamLTL}^l(\sim)$  via normal forms that also allowed us to relate the expressive power of these logics to the corresponding logics in the strict semantics. Our results show that, while the synchronous TeamLTL can be viewed as a fragment of second-order logic, the asynchronous  $\text{TeamLTL}(\otimes)$  under the lax semantics is a sublogic of HyperLTL (see [2] for a definition). Furthermore, our decidability results show, e.g, that it will probably be possible to devise a complete proof system for the logic. The full version of this article [13] relates and applies our results to recently defined logics whose asynchronicity is formalised via time evaluation functions [7]. We conclude with open questions:

- Does Theorem 14 extend to all formulae of  $\text{TeamLTL}^l(\sim)$ ? Note that any quasi-flat- $\text{TeamLTL}(\sim)$ -formula can be rewritten in HyperLTL.
- Can the result (iii) of Theorem 17 be accompanied by an matching lower bound (i.e.,  $\text{TOWER}(\text{poly})$ -hardness result)?

- Can a syntactic characterisation (similar to Corollary 11) be obtained for the downward closed fragment of  $\text{TeamLTL}^l(\sim)$ ? We believe that  $\text{TeamLTL}^l(\otimes)$  is a promising candidate, as its extensions with infinite conjunctions and disjunctions suffices for all downward closed properties of teams.
- What is the complexity of model checking for  $\text{TeamLTL}(\otimes)$  under the strict semantics?

---

## References

- 1 Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 694–717. Springer, 2021.
- 2 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST 2014*, pages 265–284, 2014.
- 3 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
- 4 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *LICS 2019*, pages 1–13. IEEE, 2019.
- 5 Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020. doi:10.1007/s00236-019-00358-2.
- 6 Pietro Galliani. Inclusion and exclusion dependencies in team semantics: On some logics of imperfect information. *Annals of Pure and Applied Logic*, 163(1):68–84, 2012.
- 7 Jens Oliver Gutsfeld, Arne Meier, Christoph Ohrem, and Jonni Virtema. Temporal team semantics revisited. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 44:1–44:13. ACM, 2022. doi:10.1145/3531130.3533360.
- 8 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021. doi:10.1145/3434319.
- 9 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of propositional logics in team semantic. *ACM Trans. Comput. Log.*, 19(1):2:1–2:14, 2018.
- 10 Lauri Hella, Antti Kuusisto, Arne Meier, and Jonni Virtema. Model checking and validity in propositional and modal inclusion logics. *J. Log. Comput.*, 29(5):605–630, 2019. doi:10.1093/logcom/exz008.
- 11 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. Modal independence logic. In Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors, *Advances in Modal Logic 10, invited and contributed papers from the tenth conference on "Advances in Modal Logic," held in Groningen, The Netherlands, August 5-8, 2014*, pages 353–372. College Publications, 2014. URL: <http://www.aiml.net/volumes/volume10/Kontinen-Mueller-Schnoor-Vollmer.pdf>.
- 12 Juha Kontinen and Max Sandström. On the expressive power of  $\text{TeamLTL}$  and first-order team logic over hyperproperties. In *WoLLIC*, volume 13038 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2021.
- 13 Juha Kontinen, Max Sandström, and Jonni Virtema. Set semantics for asynchronous  $\text{TeamLTL}$ : Expressivity and complexity. *CoRR*, abs/2304.10915, 2023. doi:10.48550/arXiv.2304.10915.
- 14 Andreas Krebs, Arne Meier, and Jonni Virtema. A team based variant of CTL. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015*, pages 140–149. IEEE Computer Society, 2015. doi:10.1109/TIME.2015.11.
- 15 Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *MFCS 2018*, volume 117, pages 10:1–10:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

- 16 Martin Lück. Axiomatizations of team logics. *Ann. Pure Appl. Log.*, 169(9):928–969, 2018.
- 17 Martin Lück. On the complexity of team logic and its two-variable fragment. In *MFCS*, volume 117 of *LIPICs*, pages 27:1–27:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 18 Martin Lück. On the complexity of linear temporal logic with team semantics. *Theoretical Computer Science*, 2020.
- 19 Martin Lück. *Team logic: axioms, expressiveness, complexity*. PhD thesis, University of Hanover, Hannover, Germany, 2020. URL: <https://www.repo.uni-hannover.de/handle/123456789/9430>.
- 20 John McLean. Proving noninterference and functional correctness using traces. *Journal of Computer Security*, 1(1):37–58, 1992. doi:10.3233/JCS-1992-1103.
- 21 Nir Piterman and Amir Pnueli. Temporal logic and fair discrete systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 27–73. Springer, 2018. doi:10.1007/978-3-319-10575-8\_2.
- 22 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977.
- 23 Markus N. Rabe. *A Temporal Logic Approach to Information-Flow Control*. PhD thesis, Saarland University, 2016.
- 24 A. W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 114–127. IEEE Computer Society, 1995. doi:10.1109/SECPRI.1995.398927.
- 25 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985. doi:10.1145/3828.3837.
- 26 Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.
- 27 Jonni Virtema. Complexity of validity for propositional dependence logics. *Inf. Comput.*, 253:224–236, 2017. doi:10.1016/j.ic.2016.07.008.
- 28 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-time temporal logic with team semantics: Expressivity and complexity. *CoRR*, abs/2010.03311, 2020. arXiv:2010.03311.
- 29 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-time temporal logic with team semantics: Expressivity and complexity. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPICs*, pages 52:1–52:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.FSTTCS.2021.52.
- 30 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003), 30 June – 2 July 2003, Pacific Grove, CA, USA*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

# Parameterized Complexity of Domination Problems Using Restricted Modular Partitions

Manuel Lafond  

Department of Computer Science, Université de Sherbrooke, Canada

Weidong Luo  

Department of Computer Science, Université de Sherbrooke, Canada

---

## Abstract

For a graph class  $\mathcal{G}$ , we define the  $\mathcal{G}$ -modular cardinality of a graph  $G$  as the minimum size of a vertex partition of  $G$  into modules that each induces a graph in  $\mathcal{G}$ . This generalizes other module-based graph parameters such as neighborhood diversity and iterated type partition. Moreover, if  $\mathcal{G}$  has bounded modular-width, the  $W[1]$ -hardness of a problem in  $\mathcal{G}$ -modular cardinality implies hardness on modular-width, clique-width, and other related parameters. Several FPT algorithms based on modular partitions compute a solution table in each module, then combine each table into a global solution. This works well when each table has a succinct representation, but as we argue, when no such representation exists, the problem is typically  $W[1]$ -hard. We illustrate these ideas on the generic  $(\alpha, \beta)$ -domination problem, which is a generalization of known domination problems such as BOUNDED DEGREE DELETION,  $k$ -DOMINATION, and  $\alpha$ -DOMINATION. We show that for graph classes  $\mathcal{G}$  that require arbitrarily large solution tables, these problems are  $W[1]$ -hard in the  $\mathcal{G}$ -modular cardinality, whereas they are fixed-parameter tractable when they admit succinct solution tables. This leads to several new positive and negative results for many domination problems parameterized by known and novel structural graph parameters such as clique-width, modular-width, and cluster-modular cardinality.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms

**Keywords and phrases** modular-width, parameterized algorithms,  $W$ -hardness,  $\mathcal{G}$ -modular cardinality

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.61

**Related Version** *Full Version*: <https://arxiv.org/abs/2307.02021> [30]

## 1 Introduction

Modular decompositions of graphs have played an important role in algorithms since their inception [23]. In the world of parameterized complexity [11, 13], Gajarský et. al. [22] proposed the notion of *modular-width*, or *mw* for short, which can be defined as the maximum degree of a prime node in the modular decomposition tree of  $G$ . Unlike other structural parameters such as treewidth [4], *mw* can be bounded on certain classes of dense graphs, making it comparable to the clique-width (*cw*) parameter [10]. In fact, *cw* is at most *mw* plus two, and *mw* can sometimes be arbitrarily larger than *cw*. It is known that several problems that are hard in *cw* are fixed-parameter tractable (FPT) in *mw*, with popular examples including HAMILTONIAN CYCLE, GRAPH COLORING [16, 18, 22], and METRIC DIMENSION [2, 5]. In particular in [22], the main technique used to design such algorithms is dynamic programming over the modular decomposition. In essence, the values of an optimal solution are found recursively in each module of the graph  $G$ , which are then combined into a solution for the graph itself, often using small integer linear programs based on the prime graph of  $G$ . Another technique was recently introduced in [19], where the authors show that the number of potential maximal cliques of a graph is at most  $O^*(1.73^{mw})$ , a fact that can be combined with results of [6, 20] to obtain FPT algorithms for a family of problems.



© Manuel Lafond and Weidong Luo;

licensed under Creative Commons License CC-BY 4.0

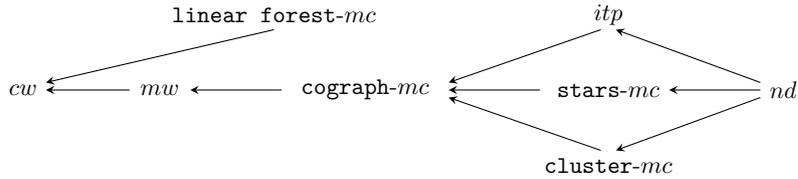
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 61; pp. 61:1–61:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Relation among the parameters clique-width ( $cw$ ), modular-width ( $mw$ ), **cograph**-modular cardinality (**cograph- $mc$** ), iterated type partitions ( $itp$ ), **stars**-modular cardinality (**stars- $mc$** ), **cluster**-modular cardinality (**cluster- $mc$** ), neighborhood diversity ( $nd$ ), and **linear forest**-modular cardinality (**linear forest- $mc$** ). Arrows indicate generalizations, e.g. modular-width( $mw$ ) generalizes **cograph**-modular cardinality and thus is bounded by (a function of) **cograph**-modular cardinality.

Despite these efforts, there are still several problems that are known to be hard on  $cw$ , for instance MAX-CUT [17] and BOUNDED DEGREE DELETION [3, 24], but unknown to be hard or FPT in  $mw$ . Recently, an XP algorithm is given for  $k$ -DOMINATION in parameter treewidth ( $tw$ ) [34], but the W-hardness for  $k$ -DOMINATION in parameter  $tw$ ,  $cw$  or  $mw$  is unknown. Also, the authors of [22] conclude with the question of whether EDGE DOMINATING SET and PARTITION INTO TRIANGLES are FPT in  $mw$ , which are 10 years-old questions that are still unanswered. One promising direction to gain knowledge and tools for  $mw$  algorithms is to study some of its related parameters. We consider such variants in which the graph must be decomposed into modules that each induces a subgraph belonging to a specific graph class. Notable examples include neighborhood diversity ( $nd$ ) [31], in which each module must induce an edgeless graph or a clique, and iterated type partition ( $itp$ ) [9], in which each module must induce a cograph. This idea was also used in [25] from which we borrow our terminology, where a partition into modules inducing cliques is used to obtain linear kernels for the cluster editing problem. Meanwhile, as Knop wrote in [29], ‘another important task in this area is to understand the boundary between modular-width on one side, and neighborhood diversity, twin-cover number, and clique-width on the other side’.

Our work resides in this boundary, as we propose to generalize the above ideas by restricting modules to a given graph class  $\mathcal{G}$ . That is, we define the  $\mathcal{G}$ -modular cardinality of a graph  $G$ , denoted  $\mathcal{G}\text{-}mc(G)$ , as the size of the smallest partition of its vertices into modules that each induces a subgraph in  $\mathcal{G}$ . If  $\mathcal{G}$  has bounded modular-width (e.g. cographs), the hardness of a problem in  $\mathcal{G}$ -modular cardinality implies its hardness in  $mw$  (and thus  $cw$ ). On the other hand, FPT techniques for  $\mathcal{G}\text{-}mc$  may shed light towards developing better algorithms for  $mw$ . Also, by considering graph classes of unbounded modular-width (e.g. paths, grids),  $\mathcal{G}\text{-}mc$  may be incomparable with  $mw$  or even  $cw$ , leading to FPT algorithms for novel types of graphs. To the best of our knowledge, such a generalization had not been studied, although it is worth mentioning that in [27], the authors propose a similar concept for treewidth, where some bags of a tree decomposition are allowed to be in some graph class.

**Our contributions.** We first establish that if  $\mathcal{G}$  is hereditary and has bounded  $mw$ , then  $mw(G)$  is at most  $\mathcal{G}\text{-}mc(G)$  for a graph  $G \notin \mathcal{G}$ , allowing the transfer of hardness results. We then show that for many graph classes, namely those that are *easily mergeable*, there is a polynomial-time algorithm to compute  $\mathcal{G}\text{-}mc$  and obtain a corresponding modular partition.

We then introduce techniques to obtain W[1]-hardness results and FPT algorithms for the  $\mathcal{G}\text{-}mc$  parameter. In essence, we argue that the dynamic programming technique on  $mw$  algorithms works well when a small amount of information from each module is sufficient to obtain a solution for the whole graph (for instance, the algorithms of [22] require only a

■ **Table 1** Our results for the  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION  $((\alpha, \beta)$ -LDD) on different parameters (the mark \* is implied by [3, 24]). Results in boldface are those proved directly in this paper (other entries are implied by these results). Recall that  $\alpha = 0$  is equivalent to the  $k$ -DOMINATION,  $\alpha = 1$  to the BOUNDED DEGREE DELETION (BDD), and  $\alpha \in (0, 1)$  to the  $\alpha$ -DOMINATION. Not shown in the table: BDD is FPT in parameters **linear forest- $mc$**  and **binary forest- $mc$** .

$(\alpha, \beta)$ -LDD problem	$\alpha = 0$ ( $k$ -dom)	$\alpha \in (0, 1)$ ( $\alpha$ -dom + $\beta$ )	$\alpha = 1$ (BDD)	Parameters
$\beta$ is in the input	W[1]-h	W[1]-h	W[1]-h*	<i>cw</i>
	W[1]-h	W[1]-h	W[1]-h	<i>mw</i>
	W[1]-h	W[1]-h	W[1]-h	<b>cograph-<math>mc</math></b>
	W[1]-h	W[1]-h	W[1]-h	<i>itp</i>
	<b>W[1]-h</b>	W[1]-h	<b>W[1]-h</b>	<b>stars-<math>mc</math></b>
	open	open	<b>FPT</b>	<b>cluster-<math>mc</math></b>
$\beta$ is any constant	<b>FPT</b>	W[1]-h	<b>FPT</b>	<i>cw</i>
	FPT	W[1]-h	FPT	<i>mw</i>
	FPT	W[1]-h	FPT	<b>cograph-<math>mc</math></b>
	FPT	W[1]-h	FPT	<i>itp</i>
	FPT	<b>W[1]-h</b>	FPT	<b>stars-<math>mc</math></b>
	FPT	open	FPT	<b>cluster-<math>mc</math></b>
	FPT	FPT	FPT	<i>nd</i>

single integer from each module). Such *succinct solution tables* from each module can often be combined using integer programs with few integer variables [21, 28, 33]. Conversely, when too much information is required from each module (e.g. linear in the size of the modules) to obtain a final solution, we are unable to use integer programming and this typically leads to W[1]-hardness. This occurs when *arbitrary solution tables* are possible in each module.

We use a large class of domination problems to illustrate these techniques. Specifically, for a real number  $\alpha \in [0, 1]$  and integer  $\beta$  (possibly negative), we introduce the  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION problem. Given a graph  $G$  and an integer  $q$ , this problem asks for a  $X \subseteq V(G)$  of size at most  $q$  such that, for each  $v \in V(G) \setminus X$ , we have  $|N(v) \cap X| \geq \alpha|N(v)| + \beta$ . In other words, each unchosen vertex has at least a fraction  $\alpha$  of its neighbors dominating it, plus some number  $\beta$ . The problem is equivalent to the BOUNDED DEGREE DELETION (BDD) [3, 24] if  $\alpha = 1$  and  $\beta \leq 0$ ; equivalent to the  $k$ -DOMINATION [8, 32] if  $\alpha = 0$  and  $\beta \geq 1$ ; and equivalent to the  $\alpha$ -DOMINATION [1, 12, 14, 35] if  $\alpha \in (0, 1)$  and  $\beta = 0$ .

Table 1 illustrates the main results of this paper. The hardness results follow from a more general result on arbitrary solution tables (Theorem 7). It is slightly technical, so we describe its high-level implication on the case  $\alpha = 1$  (BDD). In this problem, a possible solution table is a function  $f : [n] \rightarrow \mathbb{N}$  such that  $f(i)$  is the minimum maximum degree achievable in  $G$  after deleting  $i$  vertices. The theorem states that for graph class  $\mathcal{G}$ , if for any such  $f$  we can construct a graph in  $\mathcal{G}$  whose solution table is  $f$ , then BDD is W[1]-hard in  $\mathcal{G}$ -modular cardinality. We show that the class of disjoint stars satisfies this property, which implies several other hardness results. On the other hand, several positive results for the BDD make use of succinct solution tables. In essence, when  $f$  can be represented by a constant number of linear functions, or by a convex function, then we can use integer programming to merge these tables and obtain positive results. Finally, additional results not shown in the table can be deduced easily from this technique. We show that BDD is FPT in **linear forest-modular cardinality** and **binary forest-modular cardinality** as parameters, which are of interest since they are incomparable with modular-width. Due to space constraints, we refer the reader to the full version of this paper [30] for definitions and complete proofs.

## 2 Preliminary notions

For an integer  $n$ , denote  $[n] = \{1, \dots, n\}$ . The maximum degree of a graph  $G$  is denoted by  $\Delta(G)$ .  $\overline{G}$  denotes the *complement* graph of  $G$ . The neighborhood of  $v \in V(G)$  is  $N(v)$ . The set of connected components of a graph  $G$  is denoted  $CC(G)$ . For  $X \subseteq V(G)$ ,  $G[X]$  denotes the subgraph of  $G$  induced by  $X$  and  $G - X = G[V(G) \setminus X]$ . If  $X = \{v\}$ , we may write  $G - v$ . Slightly abusing notation, we may also write  $v \in G$  instead of  $v \in V(G)$ ,  $|G|$  instead of  $|V(G)|$ , and  $X \cap G$  instead of  $X \cap V(G)$ .

A *graph class*  $\mathcal{G}$  is a (possibly infinite) set of graphs containing at least one non-empty graph. We say that  $\mathcal{G}$  is *hereditary* if, for any  $G \in \mathcal{G}$ , any induced subgraph of  $G$  is also in  $\mathcal{G}$ . Note that if  $\mathcal{G}$  is hereditary, the graph consisting of an isolated vertex is in  $\mathcal{G}$ . We say that  $\mathcal{G}$  is a *polynomial-time recognition graph class* if there is a polynomial-time algorithm that decides whether a given graph  $G$  is in  $\mathcal{G}$ . Some popular graph classes that we will use throughout this paper:  $\mathcal{I}$  is the set of all edgeless graphs;  $\mathcal{K}$  is the set of all complete graphs; **cluster** is the set of graphs in which every connected component induces a complete graph; **stars** is the set of graphs in which every connected component is a star graph; **cograph** is the set of cographs, where a cograph is either a single vertex, or a graph obtained by applying either a join or a disjoint union of two cographs [7]. Observe that  $\mathcal{I} \subseteq \mathbf{stars} \subseteq \mathbf{cograph}$ .

**Modular parameters.** For a graph  $G = (V, E)$ , a *module* of  $G$  is a  $M \subseteq V$  such that for every  $v \in V \setminus M$ , either  $M \subseteq N(v)$  or  $M \cap N(v) = \emptyset$ . The empty set,  $V$ , and every  $\{v\}$  for  $v \in V$  are called the *trivial modules*. In a *prime* graph, all modules are trivial. A *factor* is a subgraph induced by a module. A module  $M$  is *strong* if for any module  $M'$  of  $G$ , either  $M' \subseteq M$ ,  $M \subseteq M'$ , or  $M \cap M' = \emptyset$ .  $M$  is *maximal* if  $M \subsetneq V$  and there is no module  $M'$  such that  $M \subsetneq M' \subsetneq V$ . A partition  $\mathcal{M}$  of  $V(G)$  is called a *modular partition* if every element of  $\mathcal{M}$  is a module of  $G$ . If  $\mathcal{M}$  only contains maximal strong modules, then it is a *maximal modular partition*. This partition is unique. Two modules  $M$  and  $M'$  are *adjacent* in  $G$  if every vertex of  $M$  is adjacent to every vertex of  $M'$ , and *non-adjacent* otherwise. For a modular partition  $\mathcal{M}$  of  $V$ , the *quotient graph*  $G_{/\mathcal{M}}$  is defined by  $V(G_{/\mathcal{M}}) = \{v_M : M \in \mathcal{M}\}$  and  $v_{M_1}v_{M_2} \in E(G_{/\mathcal{M}})$  if and only if  $M_1, M_2$  are adjacent.

We call  $MD(G)$  the *modular decomposition tree* of  $G$ , in which each vertex  $v_M$  corresponds to a strong module  $M$ . More specifically, each leaf  $v_{\{v\}}$  of the inclusion tree corresponds to a vertex  $v$  of  $G$  and the root vertex  $v_V$  corresponds to  $V$ . Moreover, for any two strong modules  $M$  and  $M'$ ,  $M'$  is a proper subset of  $M$  if and only if  $v_{M'}$  is a descendant of  $v_M$  in  $MD(G)$ . An internal vertex  $v_M$  of  $MD(G)$  is *parallel* if  $G[M]$  is disconnected, is *series* if  $\overline{G[M]}$  is disconnected, and is *prime* otherwise. The *modular-width* of  $G$  is the maximum number of children of a prime vertex in  $MD(G)$  (see [26] for more). Our variant follows.

► **Definition 1.** Let  $\mathcal{G}$  be a graph class. For a graph  $G$  (not necessarily in  $\mathcal{G}$ ), a module  $M$  of  $G$  is a  $\mathcal{G}$ -module if  $G[M]$  belongs to  $\mathcal{G}$ . A modular partition  $\mathcal{M} = \{M_1, \dots, M_k\}$  of a graph  $G$  is called a  $\mathcal{G}$ -modular partition if each  $M_i$  is a  $\mathcal{G}$ -module. The  $\mathcal{G}$ -modular cardinality of  $G$ , denoted by  $\mathcal{G}\text{-mc}(G)$ , is the minimum cardinality of a  $\mathcal{G}$ -modular partition of  $G$ .

The neighborhood diversity (*nd*) is equivalent to the  $(\mathcal{K} \cup \mathcal{I})$ -modular cardinality. The iterated type partition (*itp*) parameter [9] is the number of vertices of the graph obtained through the following process: start with the smallest modular partition into cliques and edgeless graphs; contract each module into a single vertex; repeat until no more contraction is possible. It can be shown that the remaining vertices represent modules that are cographs. Thus,  $itp(G)$  is not smaller than the **cograph**-modular cardinality of  $G$ .

### 3 Properties and tractability of $\mathcal{G}$ -modular cardinality

In this section, we state two basic results regarding  $\mathcal{G}$ - $mc$ . First,  $mw$  is not larger than  $\mathcal{G}$ - $mc$  for graph classes of bounded modular-width. This allows hardness results on  $\mathcal{G}$ - $mc$  to also apply to  $mw$ . Second,  $\mathcal{G}$ - $mc$  is polynomial-time computable for “easily mergeable” graph classes. The first is relatively easy using modular partitions and induction.

► **Theorem 2.** *Let  $\mathcal{G}$  be an hereditary graph class and define  $\omega_{\mathcal{G}} := \max_{H \in \mathcal{G}} mw(H)$ . Then for any graph  $G$ ,  $mw(G) \leq \max(\mathcal{G}\text{-}mc(G), \omega_{\mathcal{G}})$ .*

Let us note that the above bound is tight, in the sense that  $mw(G)$  can be at least as large as either  $\mathcal{G}\text{-}mc(G)$  or  $\omega(G)$ . For instance, if  $G \notin \mathcal{G}$  is a prime graph, then  $mw(G) = \mathcal{G}\text{-}mc(G)$ , and if  $G = \arg \max_{H \in \mathcal{G}} mw(H)$ , then  $mw(G) = \omega_{\mathcal{G}}$ .

Let us now state the second result. A graph  $G$  is a  $\mathcal{G}$ -join if  $\overline{G}$  is disconnected and  $G[C] \in \mathcal{G}$  for each  $C \in CC(\overline{G})$ . Likewise,  $G$  is a  $\mathcal{G}$ -union if  $G$  is disconnected and  $G[C] \in \mathcal{G}$  for each  $C \in CC(G)$ . If  $G$  is a  $\mathcal{G}$ -join (resp.  $\mathcal{G}$ -union), a  $\mathcal{G}$ -merge is a  $\mathcal{G}$ -modular partition  $\mathcal{M}$  of  $G$  such that for each  $C \in CC(\overline{G})$  (resp.  $C \in CC(G)$ ), there is some  $M \in \mathcal{M}$  that contains  $C$ . We say that  $\mathcal{M}$  is a *minimum  $\mathcal{G}$ -merge* if no other  $\mathcal{G}$ -merge has a size strictly smaller than  $\mathcal{M}$ . We say that a graph class  $\mathcal{G}$  is *easily mergeable* if there exists a polynomial time algorithm that, given a graph  $G$  such that  $G$  is either a  $\mathcal{G}$ -join or a  $\mathcal{G}$ -union, outputs a minimum  $\mathcal{G}$ -merge of  $G$ . We say that  $\mathcal{G}$  is *trivially mergeable* if, for any  $\mathcal{G}$ -join or  $\mathcal{G}$ -union  $G$ , one of  $\{V(G)\}, CC(G)$ , or  $CC(\overline{G})$  is a minimum  $\mathcal{G}$ -merge of  $G$ .

► **Theorem 3.** *Suppose that  $\mathcal{G}$  is a hereditary graph class. Suppose further that  $\mathcal{G}$  is polynomial-time recognizable and easily mergeable. Then a  $\mathcal{G}$ -modular partition of  $G$  of minimum size can be obtained in polynomial time.*

The above implies that for  $\mathcal{G} \in \{\mathcal{I}, \mathcal{K}, \text{cograph}, \text{cluster}\}$ , a  $\mathcal{G}$ -modular partition of minimum size of a graph  $G$  can be computed in polynomial time. It is not hard to show that  $\mathcal{I}, \mathcal{K}$ , and  $\text{cograph}$  are trivially mergeable. For  $\text{cluster}$ , if  $G$  is a  $\text{cluster}$ -join, then  $G \in \text{cluster}$ . If  $G$  is a  $\text{cluster}$ -union, one may show that we can merge all the connected components of  $\overline{G}$  that contain one clique into a single clique, and the other connected components are left intact.

Note that not all graph classes are equally easy to merge. Consider the class “graphs with at most 100 vertices”. Merging such graphs amounts to solving a bin packing problem with a fixed capacity of 100, and current polynomial-time algorithms require time in the order of  $n^{100}$ . This is easily mergeable nonetheless, and it would be interesting to find graph classes that are hereditary and polynomial-time recognizable, but not easily mergeable.

### 4 Hardness of domination problems with arbitrary solution tables

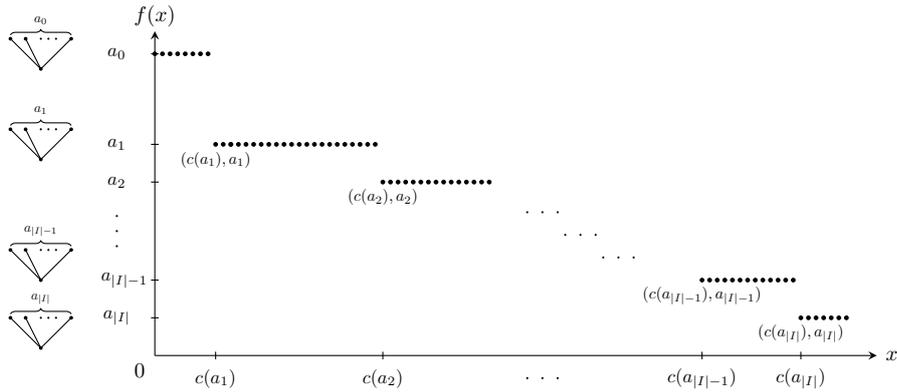
Let us recall our generic domination problem of interest. For  $\alpha \in [0, 1]$  and  $\beta \in \mathbb{Z}$  we define:

The  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION problem

**Input:** a graph  $G = (V, E)$  and a non-negative integer  $q$ ;

**Question:** does there exist a subset  $X \subseteq V$  of size at most  $q$  such that for every  $v \in V \setminus X$ ,  $|N(v) \cap X| \geq \alpha|N(v)| + \beta$ ?

In the above, the vertex set  $X$  is called a  $(\alpha, \beta)$ -linear degree dominating set of  $G$ . In addition, for convenience, we also call  $X$  the deletion part of  $G$ . For any vertex  $v \in V(G - X)$ , the degree of  $v$  in  $X$ , denoted by  $\deg(v, G, X)$ , equals the number of vertices in  $X \cap V(G)$  adjacent to  $v$ . The minimum degree of  $G - X$  in  $X$  equals  $\min\{\deg(v, G, X) : v \in G - X\}$ , which is denoted by  $\delta(G - X, X)$ .



■ **Figure 2** The degree deletion function of a  $(a_0, I, c)$ -degree deletion graph. The  $x$ -axis represents the number of vertices to delete, while the  $y$ -axis represents the minimum maximum degree achievable by deleting  $x$  vertices. The first point of the  $(i+1)$ -th horizontal vertex set is  $(c(a_i), a_i)$  for  $0 \leq i \leq |I|$ , and the last point of the  $(i+1)$ -th horizontal vertex set is  $(c(a_{i+1}) - 1, a_i)$  for  $0 \leq i \leq |I| - 1$ . Disjoint stars are used as an example here since the graph class **stars** admits arbitrary deletion tables. The number of stars with  $a_i$  leaves is  $c(a_{i+1}) - c(a_i)$  for  $0 \leq i \leq |I| - 1$ . The degree deletion process removes the internal vertices of stars from large to small.

Our goal is to formalize the intuition that graph classes with arbitrary solution tables lead to  $W[1]$ -hardness in  $\mathcal{G}$ - $mc$ . For  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION, this takes the form of *arbitrary deletion tables* for  $\alpha \in (0, 1]$  and *arbitrary retention tables* for  $\alpha = 0$ .

► **Definition 4.** Let  $(a_0, I, c)$  be a triple with  $\{a_0\} \cup I \subseteq \mathbb{N}$  and  $c : \mathbb{N} \rightarrow \mathbb{N}$ . We call  $c$  a cost function. We say that  $(a_0, I, c)$  is decreasing valid if, by listing the elements  $a_1, \dots, a_{|I|}$  of  $I$  in decreasing order, we have  $a_0 > a_1 > \dots > a_{|I|} > 1$ , and  $0 = c(a_0) < c(a_1) < \dots < c(a_{|I|})$ .

For a decreasing valid triple  $(a_0, I, c)$ , we say that a graph  $G = (V, E)$  is a  $(a_0, I, c)$ -degree deletion graph if all of the following conditions hold:

1.  $G$  has maximum degree  $a_0$  and at most  $(a_0 c(a_{|I|}))^{10}$  vertices;
2. for any  $a_i \in I$ , there exists  $X \subseteq V$  of size  $c(a_i)$  such that  $G - X$  has maximum degree  $a_i$ ;
3. for any  $a_i \in I$  and any  $X \subseteq V$  of size strictly less than  $c(a_i)$ ,  $G - X$  has maximum degree at least  $a_{i-1}$ .

In addition, if  $G \in \mathbf{stars}$  and satisfies the above three conditions, then we say that  $G$  is a  $(a_0, I, c)$ -degree deletion star graph.

We say that a graph class  $\mathcal{G}$  admits *arbitrary degree deletion tables* if, for any decreasing valid triple  $(a_0, I, c)$ , one can construct in time polynomial in  $a_0 c(a_{|I|})$  a graph  $G \in \mathcal{G}$  such that  $G$  is a  $(a_0, I, c)$ -degree deletion graph. Note that the size of  $G$  is only required to be a polynomial function of  $a_0 c(a_{|I|})$ , but we fix it to  $(a_0 c(a_{|I|}))^{10}$  for convenience. For an integer  $x \in [0, |V|]$ , we call  $f(x) = \min\{\Delta(G - X) : |X| = x\}$  the *degree deletion function* of  $G$ , where  $X$  is a subset of  $V$ . Figure 2 demonstrates the degree deletion function of a  $(a_0, I, c)$ -degree deletion graph. The intuition behind degree deletion graphs is that their deletion function has a stepwise behavior with many steps.

The above notion of an arbitrary solution table works well for  $\alpha \in (0, 1]$ . For  $\alpha = 0$ , we need to replace “deletion” with “retention”. This is useful for the  $\alpha = 0$  case, where the set  $X$  must contain at least  $\beta$  neighbors of each unchosen vertex. Hence, the steps of the table describe, for each number  $x$  of vertices to include in  $X$ , the maximum possible  $\delta(G - X, X)$  that can be achieved with a subset of size  $x$ .

► **Definition 5.** Let  $(a_0, I, c)$  be a triple with  $\{a_0\} \cup I \subseteq \mathbb{N}$  and  $c : \mathbb{N} \rightarrow \mathbb{N}$ . We call  $c$  a cost function. We say that  $(a_0, I, c)$  is increasing valid if, by listing the elements  $a_1, \dots, a_{|I|}$  of  $I$  in decreasing order, we have  $a_0 > a_1 > \dots > a_{|I|} > 100$ ,<sup>1</sup> and  $c(a_0) > c(a_1) > \dots > c(a_{|I|}) = 0$ .

For a increasing valid triple  $(a_0, I, c)$ , we say that a graph  $G = (V, E)$  is a  $(a_0, I, c)$ -degree retention graph above  $(p, l)$ , where  $p$  and  $l$  are positive integers and  $l < 100$ , if all of the following conditions hold:

1.  $G$  has maximum degree  $a_0$ , at most  $(a_0 c(a_0))^{10}$  vertices, and  $p$  vertices of degree less than  $l$ ;
2. for any  $a_i \in \{a_0\} \cup I$ , there exists  $X \subseteq V$  of size  $p + c(a_i)$  such that the minimum degree of  $G - X$  in  $X$  is  $a_i$ ;
3. for any  $a_i \in \{a_0\} \cup I$  and any  $X \subseteq V$  of size strictly less than  $p + c(a_i)$ , the minimum degree of  $G - X$  in  $X$  is at most  $a_{i+1}$ , where here we define  $a_{|I|+1} = l$ .

We say that a graph class  $\mathcal{G}$  admits arbitrary degree retention tables if, for any increasing valid triple  $(a_0, I, c)$ , there exist integers  $p$  and  $l$  such that one can construct in time polynomial in  $c(a_0)a_0$  a graph  $G \in \mathcal{G}$  such that  $G$  is a  $(a_0, I, c)$ -degree retention graph above  $(p, l)$ . Note that condition 2 of Definition 5 imply that  $p + c(a_0) < |V| \leq (c(a_0)a_0)^{10}$ . For any integer  $x \in [0, |V| - 1]$ , we call  $f(x) = \max\{\delta(G - X, X) : |X| = x\}$  the degree retention function of  $G$ , where  $X$  is a subset of  $V$ . Importantly, **stars** admit both types of tables.

► **Proposition 6.** The graph class **stars** admits arbitrary deletion tables and arbitrary retention tables.

► **Theorem 7.** The  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION problem is  $W[1]$ -hard in the following cases:

1.  $\alpha = 0$ ,  $\beta$  is in the input, and the parameter is the  $(\mathcal{G} \cup \mathcal{I})$ -modular cardinality, where  $\mathcal{G}$  is any graph class that admits arbitrary degree retention tables;
2.  $\alpha$  is any fixed constant in the interval  $(0, 1)$ ,  $\beta$  is any fixed constant in  $\mathbb{Z}$ , and the parameter is the **stars**-modular cardinality;
3.  $\alpha = 1$ ,  $\beta$  is in the input, and the parameter is the  $(\mathcal{G} \cup \mathcal{I})$ -modular cardinality, where  $\mathcal{G}$  is any graph class that admits arbitrary degree deletion tables.

The proof of the Theorem 7 is quite long and can be found in [30]. The proof of the three cases all use the same construction and the same set of claims, but most claims require an argument for each case. The next section illustrates the main techniques on the case  $\alpha = 1$ . Using the relationship demonstrated in Figure 1 we have the following.

► **Corollary 8.** The  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION problem parameterized by either clique-width, modular-width, **cograph**-modular cardinality, iterated type partition, and **stars**-modular cardinality, is  $W[1]$ -hard in the following cases:

1.  $\alpha = 0$ ,  $\beta$  is in the input;
2.  $\alpha$  is any fixed constant in the interval  $(0, 1)$ ,  $\beta$  is any fixed constant in  $\mathbb{Z}$ ;
3.  $\alpha = 1$ ,  $\beta$  is in the input.

In particular, **BOUNDED DEGREE DELETION** problem,  $k$ -**DOMINATION** problem, and  $\alpha$ -**DOMINATION** problem are  $W[1]$ -hard in all these parameters.

One can show that these problems are in XP parameterized by **stars**-modular cardinality. Indeed, one can choose, for each **stars** module  $M$ , a number in  $[|M|] \cup \{0\}$  of vertices to delete in  $M$ , and apply these deletions greedily. There are at most  $|V(G)|^{\mathcal{G}\text{-}mc(G)}$  combinations

<sup>1</sup> In fact, any large constant here is enough for our proof in this paper, we fix it to be 100 for convenience.

of choices, and it suffices to try each of them. This works because the greedy strategy is applicable to stars, and it remains to study the XP complexity for other parameters. The next proposition is not related to the above hardness results, but allows us to fill in some of the gaps that the above leaves in our results table. In section 6, we will also show that for  $\alpha = 1$  and  $\beta$  in the input, the problem is FPT in `cluster`-modular cardinality.

► **Proposition 9.** *The  $(\alpha, \beta)$ -LINEAR DEGREE DOMINATION problem is FPT in the following cases:*

1.  $\alpha \in [0, 1]$ ,  $\beta$  is in the input, and the parameter is the neighborhood diversity;
2.  $\alpha \in \{0, 1\}$ ,  $\beta$  is a constant, and the parameter is the clique-width.

The first case requires some work, whereas the second is a simple application of Courcelle's theorem [10], as the problem admits a constant-length MSO<sub>1</sub> formula.

## 5 $(1, \beta)$ -Linear Degree Domination (BDD)

We sketch the proof of the W[1]-hardness of  $(1, \beta)$ -LINEAR DEGREE DOMINATION problem, which is enough to demonstrate the main idea of the reduction technique. Recall that we assume that  $\alpha = 1$  and  $\beta < 0$ , which is the BOUNDED DEGREE DELETION problem. That is, we must delete at most  $q$  vertices such that the resulting subgraph has maximum degree at most  $|\beta|$ . Furthermore, in the  $(1, \beta)$ -LINEAR DEGREE DOMINATION problem, we also call  $X$ , the  $(1, \beta)$ -linear degree dominating set of  $G$ , the deletion part of  $G$ .

We provide a reduction from the SYMMETRIC MULTICOLORED CLIQUE problem, which we define as follows. A symmetric multicolored graph  $G = (V^1 \cup V^2 \dots \cup V^k, E)$  is a connected graph such that, for all distinct  $i, j \in [k]$ ,

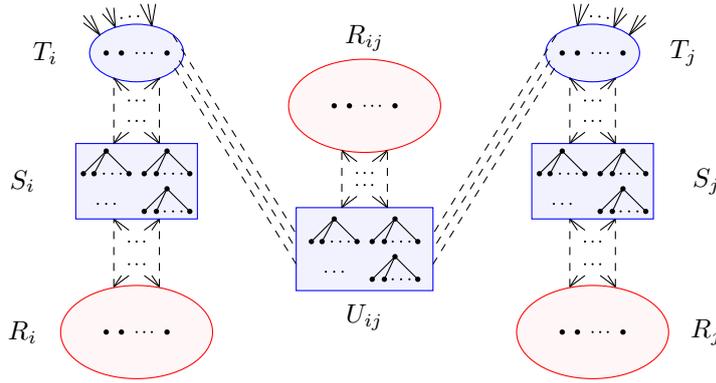
1.  $V^i = \{v_1^i, \dots, v_n^i\}$ , where  $n \geq k$ ;
2. all the vertices of  $V^i$  are colored by color  $i$ ;
3. if  $v_r^i v_s^j \in E(G)$ , then  $v_s^j v_r^i \in E(G)$  as well.

Then, for the SYMMETRIC MULTICOLORED CLIQUE problem, the input is a symmetry multicolored graph  $G$  and an integer  $k$ , and the objective is to decide whether  $G$  contains a  $k$ -clique with vertices of all  $k$  colors. We also call  $v_r^i v_s^j$  and  $v_s^i v_r^j$  symmetry edges.

The reduction in [15, Lemma 1], which proves the W[1]-hardness of the multicolored clique problem, actually produces a symmetric multicolored graph. Hence, SYMMETRIC MULTICOLORED CLIQUE is W[1]-hard. We now sketch the following.

► **Lemma 10.** *Case 3 for the W[1]-hardness results in Theorem 7 is correct.*

Let  $(G, k)$  be an instance of SYMMETRIC MULTICOLORED CLIQUE, where  $G = (V^1 \cup V^2 \cup \dots \cup V^k, E)$ . Without loss of generality, suppose  $k \geq 100$ , otherwise, the problem can be solved in polynomial time. We will construct a corresponding instance  $(H, \beta, q)$  of  $(1, \beta)$ -LINEAR DEGREE DOMINATION, where  $H$  is a graph whose  $\mathcal{G}$ -modular cardinality will be bounded by  $O(k^2)$ ,  $\beta = -(nk)^{10000}$ , and  $q$  is the maximum allowed size of  $X$ , the desired deletion part of  $H$  (to be specified later). Before proceeding, we will make use of a 2-sumfree-set, which is a set of positive integers in which every couple of elements has a distinct sum. That is,  $I$  is a 2-sumfree set if, for any  $(a, b), (a', b') \in I \times I$ ,  $a + b = a' + b'$  if and only if  $\{a, b\} = \{a', b'\}$  (note that  $a = b$  is possible). It is known that one can construct in time  $O(n^3)$  a 2-sumfree-set  $I$  of cardinality  $n$  in which the maximum value is  $n^4$ , which can be achieved with a greedy procedure (this is because  $(n+1)^4 - n^4 > n^3$  and  $a_i + a_j - a_r$  has at most  $n^3$  different values). We thus assume that we have built a 2-sumfree set  $I = \{a_1, \dots, a_n\}$  for  $H$ , where  $n^4 \geq a_1 > \dots > a_n \geq 1$ . Without loss of generality, we may multiply each  $a_i \in I$  by an



■ **Figure 3** Illustration of the construction of  $H$ .

integer  $r$ , where here we choose  $r = 2(k-1)^2k^3$ . Then, we have that  $n^4r \geq a_1 > \dots > a_n \geq r$  for the updated  $I$ . Moreover, for any distinct pair  $(a, b), (a', b') \in I \times I$ , we have that the absolute value of  $a + b - a' - b'$  is at least  $r$ .

For each color class  $V^i = \{v_1^i, \dots, v_n^i\}$  of  $G$ , we provide a bijection  $f_i$  from  $V^i$  to  $I$ , such that  $f_i(v_s^i) = a_s$  for every  $s \in [n]$ . Clearly, we can use  $f_i^{-1}(a_s)$  to denote the unique vertex  $v_s^i$  of  $V^i$  associated with  $a_s \in I$ . We have that, for all  $s, t \in [n]$ , each pair of  $a_s, a_t \in I$  has a unique sum. For distinct  $i, j$  and any  $u \in V^i, w \in V^j$  such that  $uw \in E(G)$ , if  $f_i(u) + f_j(w) = a_s + a_t$ , then edge  $uw$  is either  $v_s^i v_t^j$  or  $v_t^i v_s^j$ . Moreover, for any distinct color classes  $V^i, V^j$ , edges  $v_s^i v_t^j$  and  $v_t^i v_s^j$ , together, are both in  $E(G)$  or both not in  $E(G)$ . Hence, by looking at a sum in  $I$ , we will be able to tell whether it is corresponding to a pair of symmetry edges between  $V^i$  and  $V^j$ , or not.

Next, we define  $s = n^{10}$ ,  $q = ks + \binom{k}{2}s$ ,  $a_0 = q + 1$ , and  $a_{n+1} = a_n - 1$ . We construct  $H$  as in Figure 3. First, for each color class  $V^i$  of  $G$ , add three factors  $R_i, S_i, T_i$  to  $H$ , where:

- $R_i$  is an edgeless graph of size  $|\beta| - s$ .
- $S_i$  is a  $(a_0, I \cup \{a_{n+1}\}, c)$ -degree deletion graph, where we put the costs  $c(a_j) = s - \frac{1}{2}a_j$  for  $a_j \in I$  and  $c(a_{n+1}) = a_0$ .
- $T_i$  is an edgeless graph of size  $s$ .

We then make  $S_i$  adjacent with  $R_i$  and  $T_i$ . Secondly, for each pair of color classes  $V^i, V^j$  with  $i < j$ , we add another two factors  $U_{ij}, R_{ij}$ , where:

- $R_{ij}$  is an edgeless graph of size  $|\beta| - 2s$ .
- $U_{ij}$  is built as follows. Suppose integer set  $I_{ij}$  consists of all  $a + b$  such that  $a, b \in I$  and symmetry edges  $f_i^{-1}(b)f_j^{-1}(a), f_i^{-1}(a)f_j^{-1}(b) \in E(G)$ . Let  $\ell_{ij} = \min(I_{ij})$ .

Then  $U_{ij}$  is a  $(a_0, I_{ij} \cup \{\ell_{ij} - 1\}, c_{ij})$ -degree deletion graph, where we put the cost  $c_{ij}(a + b) = s - \frac{1}{2(k-1)}(a + b)$ , and we put  $c_{ij}(\ell_{ij} - 1) = a_0$ .

We then make  $U_{ij}$  adjacent with  $R_{ij}$ , and adjacent with  $T_i$  and  $T_j$ . To avoid cumbersome notation, we define  $U_{ij} = U_{ji}$ ,  $R_{ij} = R_{ji}$ ,  $c_{ij} = c_{ji}$ , and  $I_{ij} = I_{ji}$ . This completes the construction of  $H$ . It is easy to see that  $H$  can be constructed in polynomial time and the number of factors in  $H$  is a polynomial function in  $k$ .

All that remains is to argue that the objective instance is equivalent to the original one. The following lemma gives the forward direction. We only provide a sketch here.

► **Lemma 11.** *Suppose that  $G$  contains a multicolored clique  $C$  of size  $k$ . Then there is  $X \subseteq V(H)$  of size at most  $q = ks + \binom{k}{2}s$  such that  $\Delta(H - X) \leq |\beta|$ .*

**Proof sketch.** For  $i \in [k]$ , let  $f_i^{-1}(\hat{a}_i)$  be the vertex of  $V^i$  that belongs to the multicolored clique  $C$ , where  $\hat{a}_i \in I$  is the number associated with the vertex. For any  $i \neq j$ , we know that vertices  $f_i^{-1}(\hat{a}_i)$  and  $f_j^{-1}(\hat{a}_j)$  are in  $C$ , which means that  $f_i^{-1}(\hat{a}_i)f_j^{-1}(\hat{a}_j) \in E(G)$ . This

## 61:10 Parameterized Complexity of Domination Problems

implies that  $\hat{a}_i + \hat{a}_j$  is in the  $I_{ij}$  list that was used to construct  $U_{ij}$ . We then show how to construct the vertex set  $X$  for  $H$ . The intersection of each  $R_i$  and  $X$  is empty. For each  $T_i$ , add  $\hat{a}_i$  vertices to  $X$ . For each  $S_i$ , add  $c(\hat{a}_i) = s - \frac{1}{2}\hat{a}_i$  vertices to  $X$ . This can be done so that  $S_i - X$  has maximum degree  $\hat{a}_i$  according to Definition 4. The intersection of each  $R_{ij}$  and  $X$  is empty. For each  $U_{ij}$ , we can add  $c_{ij}(\hat{a}_i + \hat{a}_j) = s - \frac{\hat{a}_i + \hat{a}_j}{2(k-1)}$  vertices to  $X$  so that  $U_{ij} - X$  has maximum degree  $\hat{a}_i + \hat{a}_j$  according to Definition 4. It is not hard to verify that  $X$  with exactly  $q$  vertices satisfies that  $\Delta(H - X) \leq |\beta|$ . ◀

The converse direction is much more difficult.

► **Lemma 12.** *Suppose that there is  $X \subseteq V(H)$  with  $|X| \leq q$  such that  $H - X$  has maximum degree at most  $|\beta|$ . Then  $G$  contains a multicolored clique of size  $k$ .*

**Proof sketch.** Let  $X \subseteq V(H)$  be of size at most  $q$  such that  $H - X$  has maximum degree  $|\beta|$  or less. To ease notation slightly, for a factor  $M$  of  $H$ , we will write  $X(M) := X \cap V(M)$  and  $\chi(M) := |X(M)|$ . The proof is divided into a series of claims. ◀

▷ **Claim 13.** For each  $i \in [k]$ , we may assume that  $\chi(R_i) = 0$ . Moreover for each distinct  $i, j \in [k]$ , we may assume that  $\chi(R_{ij}) = 0$ .

The rough idea is that a vertex of  $X(R_i)$  can always be replaced by a vertex of  $T_i \setminus X$  if the latter is non-empty. If  $V(T_i) \setminus X$  is empty, then after an unavoidable at least  $c(a_1)$  vertices deletion from  $S_i$ , no deletion in  $R_i$  is needed since each remaining vertex of  $S_i$  in  $H - X$  has maximum degree at most  $|\beta| - s + a_1 < |\beta|$ . The idea for  $\chi(R_{ij}) = 0$  goes the same way.

▷ **Claim 14.** For any  $i \in [k]$ , we may assume  $\Delta(S_i - X) \in \{a_1, \dots, a_n\}$ , and that  $\chi(S_i) = c(\Delta(S_i - X))$ .

The rough idea is that Case 2 of Definition 4 states that we can delete  $c(a_j)$  vertices from  $S_i$  to make  $\Delta(S_i - X) = a_j$ , where  $0 \leq j \leq n + 1$ . In fact, this is the smallest maximum degree we can achieve in  $S_i$  by deleting between  $c(a_j)$  and  $c(a_{j+1}) - 1$  vertices, because of the stepwise behavior of arbitrary deletion tables. Moreover, we already know that  $\chi(R_i) = 0$  based on Claim 13. So, if  $\Delta(S_i - X) = a_0$  there is a vertex in  $S_i - X$  with degree  $|\beta| - s + a_0 > |\beta|$ , and if  $\Delta(S_i - X) = a_n - 1$  or less then  $c(a_n - 1) = a_0 > q$  vertices have to be deleted from  $S_i$ .

▷ **Claim 15.** For any distinct  $i, j \in [k]$ , we may assume that  $\Delta(U_{ij} - X) \in I_{ij}$ , and that  $\chi(U_{ij}) = c_{ij}(\Delta(U_{ij} - X))$ .

The idea for proving this claim is similar to that of Claim 14. Our next step is to argue that the degree chosen by  $U_{ij}$  must be the sum of the degrees chosen by  $S_i$  and  $S_j$ . More specifically, for  $i \in [k]$ , we say that  $S_i$  chose  $a_j \in I$  if  $\Delta(S_i - X) = a_j$  and  $\chi(S_i) = c(a_j)$ . Likewise, for distinct  $i, j \in [k]$ , we say that  $U_{ij}$  chose  $a, b \in I$  if  $\Delta(U_{ij} - X) = a + b$  and  $\chi(U_{ij}) = c_{ij}(a + b)$ . Note that by Claim 14, each  $S_i$  chooses one  $a_j$  and by Claim 15, each  $U_{ij}$  chooses one pair  $a, b$  such that symmetry edges  $f_i^{-1}(b)f_j^{-1}(a), f_i^{-1}(a)f_j^{-1}(b) \in E(G)$ . The point to make is that if  $S_i$  and  $S_j$  chose  $a$  and  $b$ , respectively, then  $U_{ij}$  must have chosen  $a, b$ .

▷ **Claim 16.** For each  $i \neq j$ , if  $S_i$  chose  $a \in I$  and  $S_j$  chose  $b \in I$ , then  $U_{ij}$  chose  $a, b$ .

We sketch it as follows. For each  $i \in [k]$ , we will denote by  $\hat{a}_i$  the element of  $I$  that  $S_i$  chose. We divide the  $U_{ij}$ 's into three groups:

$$U^< = \{U_{ij} : U_{ij} \text{ chose } a', b' \text{ such that } a' + b' < \hat{a}_i + \hat{a}_j\}$$

$$U^= = \{U_{ij} : U_{ij} \text{ chose } a', b' \text{ such that } a' + b' = \hat{a}_i + \hat{a}_j\}$$

$$U^> = \{U_{ij} : U_{ij} \text{ chose } a', b' \text{ such that } a' + b' > \hat{a}_i + \hat{a}_j\}$$

To prove the claim, it suffices to show that  $U^<$  and  $U^>$  are empty (this is because  $U_{ij} \in U^=$  is only possible if  $U_{ij}$  chose  $\hat{a}_i, \hat{a}_j$ , since all the sum pairs are distinct). The rough idea is as follows. If each  $U_{ij}$  chose the correct  $\hat{a}_i, \hat{a}_j$ , then each of them will incur a deletion cost of  $c_{ij}(\hat{a}_i + \hat{a}_j) = s - \frac{\hat{a}_i + \hat{a}_j}{2^{(k-1)}}$  and end up cancelling the deletion costs of the  $S_i$  and  $T_i$  factors. If  $U^<$  is non-empty, it incurs extra deletion cost with respect to  $c_{ij}(\hat{a}_i + \hat{a}_j)$  with no real benefit. The complicated case is when  $U^>$  is non-empty. In this case,  $U_{ij} - X$  has higher degree than if it had chosen  $\hat{a}_i, \hat{a}_j$  and incurs less deletions than  $c_{ij}(\hat{a}_i + \hat{a}_j)$ . However, this needs to be compensated with extra deletions in  $T_i$  and  $T_j$ . By using a charging argument, we can show that the sum of extra deletions required for all the  $U^>$  members outweighs the deletions saved in the  $U_{ij}$ 's of  $U^>$ .

We can now construct a multicolored clique. Define  $C = \{f_i^{-1}(\hat{a}_i) : i \in [k] \text{ and } S_i \text{ chose } \hat{a}_i\}$ . We claim that  $C$  is a clique. By Claim 14, each  $S_i$  chooses some  $\hat{a}_i$  and thus  $|C| = k$ . Now let  $f_i^{-1}(\hat{a}_i), f_j^{-1}(\hat{a}_j)$  be two vertices of  $C$ , where  $i < j$ . Then  $\hat{a}_i, \hat{a}_j$  were chosen by  $S_i$  and  $S_j$ , respectively, and by Claim 15 we know that  $U_{ij}$  chose  $\hat{a}_i + \hat{a}_j$ . By the construction of the  $U_{ij}$  solution table, this is only possible if symmetry edges  $f_i^{-1}(\hat{a}_i)f_j^{-1}(\hat{a}_j), f_i^{-1}(\hat{a}_j)f_j^{-1}(\hat{a}_i) \in E(G)$ . Therefore,  $f_i^{-1}(\hat{a}_i)f_j^{-1}(\hat{a}_j) \in E(G)$  and  $C$  is a clique.

## 6 FPT algorithms for succinct solution tables

In this section, we study the BOUNDED DEGREE DELETION problem (i.e.  $\alpha = 1, \beta < 0$ ) on graph classes that admit succinct solution tables. Let  $G = (V, E)$  be a graph. Assume  $\mathbb{S}_x \subseteq 2^V$  consists of all subsets of  $V$  with size  $x \in [0, |V|]$ .  $X \in \mathbb{S}_x$  is called an  $x$ -deletion set of  $G$  if  $\Delta(G - X) = \min\{\Delta(G - Y) : Y \in \mathbb{S}_x\}$ . An  $x$ -deletion of  $G$  is the process of deleting all vertices of an  $x$ -deletion set from  $G$ . Clearly, the degree deletion function  $f(x)$  of  $G$  is the maximum degree of  $G$  after an  $x$ -deletion. A *piecewise linear function*  $g : \mathbb{R} \rightarrow \mathbb{R}$  is a continuous function defined on a sequence of intervals, such that the function is linearly restricted to each of the intervals (each such linear function is called a *sub-function* of  $g$ ). In addition, a *constant piecewise linear function* is a piecewise linear function that consists of a constant number of linear sub-functions.

► **Definition 17.** Let  $\mathcal{G}$  be a polynomial-time recognizable graph class. For  $G \in \mathcal{G}$ , suppose that  $f^G(x)$  is the degree deletion function of  $G = (V, E)$ , where  $x \in [0, |V|]$ . We say that  $\mathcal{G}$  admits a succinct solution table for BOUNDED DEGREE DELETION if, for every  $G \in \mathcal{G}$ , there exists a function  $g^G : \mathbb{R} \rightarrow \mathbb{R}$  that satisfies at least one of the following conditions:

1.  $g^G$  is a constant piecewise linear function such that  $f^G(x) = g^G(x)$  for every integer  $x \in [0, |V(G)|]$ ,
2.  $g^G$  is a piecewise convex linear function such that  $f^G(x) = \lceil g^G(x) \rceil$  for every integer  $x \in [0, |V(G)|]$ .

Moreover,  $g^G$  can be described and constructed in polynomial time with respect to  $|V(G)|$ .

A graph class admits a succinct solution table of type  $t$  for BOUNDED DEGREE DELETION if we refer to the condition  $t$ , where  $t \in \{1, 2\}$ . Type 1 implies the solution table can be divided into a constant number of blocks, each of which can be encoded into a small number of bits

## 61:12 Parameterized Complexity of Domination Problems

(note that for any fixed graph class, the upper bound on the constant should be fixed as well). In type 2, the solution table could be convex, but could also be *non-convex* with a larger number of blocks, but we can reduce this special non-convex function to a convex function using ceilings (this occurs with `cluster-mc`). As we show, these definitions capture the intuition of solution tables that can be merged in FPT time.

► **Theorem 18.** *Let  $\mathcal{G}$  be a graph class that admits a succinct solution table for BOUNDED DEGREE DELETION. Assume a minimum  $\mathcal{G}$ -modular partition is given. Then, BOUNDED DEGREE DELETION is FPT parameterized by  $\mathcal{G}$ -modular cardinality.*

**Application.** Let graph class BDT include all graphs with bounded degree and treewidth. We prove BDT and `cluster` admit succinct solution tables of type 1 and type 2, respectively. Thus, BOUNDED DEGREE DELETION is FPT in parameter *BDT-mc* or *cluster-mc*. Clearly, *BDT-mc* is bounded by a function of vertex cover number and is incomparable with *mw* (*linear forest* has unbounded *mw*). Moreover, `cluster-mc` is a parameter with size at most *nd* and at least *mw*.

► **Theorem 19.** *BDT admits a succinct solution table for BOUNDED DEGREE DELETION. Therefore, the BOUNDED DEGREE DELETION is FPT in parameter *BDT-mc*.*

**Proof sketch.** For every  $G \in \text{BDT}$ , the degree deletion function of  $G$  can be represented by an efficiently computable step function with a constant number of steps. ◀

► **Lemma 20.** *Assume cluster graph  $H$  contains  $b$  complete graphs  $K_a$ , where  $K_a$  is the maximum size complete graph in  $H$ . Let  $q \in [b, |V(H)|]$ . Suppose  $R$  is obtained from  $H$  by deleting exactly one vertex from every  $K_a$  of  $H$ . Then,  $H$  has a  $q$ -deletion such that the maximum degree of the remaining graph is  $h$  if and only if  $R$  has a  $(q - b)$ -deletion such that the maximum degree of the remaining graph is  $h$ .*

► **Theorem 21.** *Cluster admits a succinct solution table for BOUNDED DEGREE DELETION. Therefore, the BOUNDED DEGREE DELETION is FPT in parameter *cluster-mc*.*

**Proof sketch.** Let  $G$  be a cluster graph and  $f^G(x)$  be the degree deletion function of  $G$ . We can construct a piecewise convex linear function  $g^G$  based on the structure of  $G$ , and prove that  $f^G(x) = \lceil g^G(x) \rceil$  by using the properties of the two functions and Lemma 20. ◀

**Open problems.** We conclude with some interesting problems.

- Can we characterize graph classes that are easily mergeable? For instance, is the class of  $H$ -free graphs easily mergeable, for any fixed graph  $H$ ?
- Is BOUNDED DEGREE DELETION fixed-parameter tractable in parameter  $(K_{1,t}$ -free)-modular cardinality, where  $t \geq 3$  is either fixed or a parameter?
- Is  $k$ -DOMINATION FPT in parameter `cluster`-modular cardinality, where  $\beta$  is related to the input size?
- Is  $\alpha$ -DOMINATION FPT in parameter `cluster`-modular cardinality?
- The RED-BLUE CAPACITATED DOMINATING SET problem is W[1]-hard in *cw* [17]. It is not hard to prove it to be FPT in *mw* using succinct solution tables. Does the same hold for the RED-BLUE EXACT SATURATED CAPACITATED DOMINATING SET?
- Are EDGE DOMINATING SET, MAX-CUT, and PARTITION INTO TRIANGLES FPT in parameter `cograph`-modular cardinality?

---

**References**

---

- 1 Davood Bakhshesha, Mohammad Farshia, and Mahdieh Hasheminezhada. A generalization of  $\alpha$ -dominating set and its complexity. In *The 46 th Annual Iranian Mathematics Conference*, page 753, 2015.
- 2 Rémy Belmonte, Fedor V. Fomin, Petr A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM J. Discret. Math.*, 31(2):1217–1243, 2017. doi:10.1137/16M1057383.
- 3 Nadja Betzler, Robert Brederick, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discret. Appl. Math.*, 160(1-2):53–60, 2012. doi:10.1016/j.dam.2011.08.013.
- 4 Hans L. Bodlaender. Treewidth: Structure and algorithms. In Giuseppe Prencipe and Shmuel Zaks, editors, *Structural Information and Communication Complexity, 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007, Proceedings*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2007. doi:10.1007/978-3-540-72951-8\_3.
- 5 Édouard Bonnet and Nidhi Purohit. Metric dimension parameterized by treewidth. *Algorithmica*, 83(8):2606–2633, 2021. doi:10.1007/s00453-021-00808-9.
- 6 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.
- 7 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.
- 8 Mustapha Chellali, Odile Favaron, Adriana Hansberg, and Lutz Volkmann.  $k$ -domination and  $k$ -independence in graphs: A survey. *Graphs Comb.*, 28(1):1–55, 2012. doi:10.1007/s00373-011-1040-3.
- 9 Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno. Iterated type partitions. In Leszek Gasieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, volume 12126 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2020. doi:10.1007/978-3-030-48966-3\_15.
- 10 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.
- 11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 12 F. Dahme, Dieter Rautenbach, and Lutz Volkmann. Some remarks on alpha-domination. *Discuss. Math. Graph Theory*, 24(3):423–430, 2004. doi:10.7151/dmgt.1241.
- 13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 14 Jean E. Dunbar, Dean G. Hoffman, Renu C. Laskar, and Lisa R. Markus.  $\alpha$ -domination. *Discret. Math.*, 211:11–26, 2000. doi:10.1016/S0012-365X(99)00131-4.
- 15 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.
- 16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.
- 17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

- 18 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.
- 19 Fedor V Fomin, Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Algorithms parameterized by vertex cover and modular width, through potential maximal cliques. *Algorithmica*, 80:1146–1169, 2018.
- 20 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.
- 21 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Comb.*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 22 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8\_15.
- 23 T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18(1-2):25–66, March 1967. doi:10.1007/bf02020961.
- 24 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/s00453-020-00758-8.
- 25 Jiong Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.
- 26 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.*, 4(1):41–59, 2010. doi:10.1016/j.cosrev.2010.01.001.
- 27 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.
- 28 Ravi Kannan. Minkowski’s convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.
- 29 Dusan Knop. Partitioning graphs into induced subgraphs. *Discret. Appl. Math.*, 272:31–42, 2020. doi:10.1016/j.dam.2019.01.010.
- 30 Manuel Lafond and Weidong Luo. Parameterized complexity of domination problems using restricted modular partitions. *arXiv*, 2023. arXiv:2307.02021.
- 31 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.
- 32 James K. Lan and Gerard Jennhwa Chang. Algorithmic aspects of the  $k$ -domination problem in graphs. *Discret. Appl. Math.*, 161(10-11):1513–1520, 2013. doi:10.1016/j.dam.2013.01.015.
- 33 Hendrik W. Lenstra-Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.
- 34 Ke Liu and Mei Lu.  $w$ -dominating set problem on graphs of bounded treewidth. *arXiv*, 2021. arXiv:2101.02867.
- 35 Leila Saadi, Badreddine Benreguia, Chafik Arar, and Hamouma Moumen. Self-stabilizing algorithm for minimal  $(\alpha, \beta)$ -dominating set. *Int. J. Comput. Math. Comput. Syst. Theory*, 7(2):81–94, 2022. doi:10.1080/23799927.2022.2072400.

# Parameterized Max Min Feedback Vertex Set

Michael Lampis  

Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

Nikolaos Melissinos  

Department of Theoretical Computer Science, Faculty of Information Technology,  
Czech Technical University in Prague, Czech Republic

Manolis Vasilakis  

Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

---

## Abstract

Given a graph  $G$  and an integer  $k$ , MAX MIN FVS asks whether there exists a minimal set of vertices of size at least  $k$  whose deletion destroys all cycles. We present several results that improve upon the state of the art of the parameterized complexity of this problem with respect to both structural and natural parameters.

Using standard DP techniques, we first present an algorithm of time  $\text{tw}^{O(\text{tw})}n^{O(1)}$ , significantly generalizing a recent algorithm of Gaikwad et al. of time  $\text{vc}^{O(\text{vc})}n^{O(1)}$ , where  $\text{tw}, \text{vc}$  denote the input graph's treewidth and vertex cover respectively. Subsequently, we show that both of these algorithms are essentially optimal, since a  $\text{vc}^{o(\text{vc})}n^{O(1)}$  algorithm would refute the ETH.

With respect to the natural parameter  $k$ , the aforementioned recent work by Gaikwad et al. claimed an FPT branching algorithm with complexity  $10^k n^{O(1)}$ . We point out that this algorithm is incorrect and present a branching algorithm of complexity  $9.34^k n^{O(1)}$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms

**Keywords and phrases** ETH, Feedback vertex set, Parameterized algorithms, Treewidth

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.62

**Related Version** *Full Version*: <https://arxiv.org/abs/2302.09604>

**Funding** This work is partially supported by ANR projects ANR-21-CE48-0022 (S-EX-AP-PE-AL) and ANR-18-CE40-0025-01 (ASSK).

*Nikolaos Melissinos*: Supported by the CTU Global postdoc fellowship program.

**Acknowledgements** Work primarily conducted while Nikolaos Melissinos was affiliated with Université Paris-Dauphine.

## 1 Introduction

We consider a MaxMin version of the well-studied feedback vertex set problem where, given a graph  $G = (V, E)$  and a target size  $k$ , we are asked to find a set of vertices  $S$  with the following properties: (i) every cycle of  $G$  contains a vertex of  $S$ , that is,  $S$  is a feedback vertex set (ii) no proper subset of  $S$  is a feedback vertex set, that is,  $S$  is minimal (iii)  $|S| \geq k$ . Although much less studied than its minimization cousin, MAX MIN FVS has recently attracted attention in the literature as part of a broader study of MaxMin versions of standard problems, such as MAX MIN VERTEX COVER and UPPER DOMINATING SET. The main motivation of this line of research is the search for a deeper understanding of the performance of simple greedy algorithms: given an input, we would like to compute what is the worst possible solution that would still not be improvable by a simple heuristic, such as removing redundant vertices. Nevertheless, over recent years MaxMin problems have been found to possess an interesting combinatorial structure of their own and have now become an object of more widespread study (we survey some such results below).



© Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 62; pp. 62:1–62:15

Leibniz International Proceedings in Informatics



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

It is not surprising that MAX MIN FVS is known to be NP-complete and is in fact significantly harder than MINIMUM FVS in most respects, such as its approximability or its amenability to algorithms solving special cases. Given the problem’s hardness, in this paper we focus on the parameterized complexity of MAX MIN FVS, since parameterized complexity is one of the main tools for dealing with computational intractability<sup>1</sup>. We consider two types of parameterizations: the natural parameter  $k$ ; and the parameterization by structural width measures, such as treewidth. In order to place our results into perspective, we first recall the current state of the art.

**Previous work.** MAX MIN FVS was first shown to be NP-complete even on graphs of maximum degree 9 by Mishra and Sikdar [32]. This was subsequently improved to NP-completeness for graphs of maximum degree 6 by Dublois et al. [20], who also present an approximation algorithm with ratio  $n^{2/3}$  and proved that this is optimal unless  $P=NP$ . A consequence of the polynomial time approximation algorithm of [20] was the existence of a kernel of order  $O(k^3)$ , which implied that the problem is fixed-parameter tractable with respect to the natural parameter  $k$ . Some evidence that this kernel size may be optimal was later given by [2]. We note also that the problem can easily be seen to be FPT parameterized by treewidth (indeed even by clique-width) as the property that a set is a minimal feedback vertex set is  $MSO_1$ -expressible, so standard algorithmic meta-theorems apply.

Given the above, the state of the art until recently was that this problem was known to be FPT for the two most well-studied parameterizations (by  $k$  and by treewidth), but concrete FPT algorithms were missing. An attempt to advance this state of the art and systematically study the parameterized complexity of the problem was recently undertaken by Gaikwad et al. [23], who presented exact algorithms for this problem running in time  $10^k n^{O(1)}$  and  $vc^{O(vc)} n^{O(1)}$ , where  $vc$  is the input graph’s vertex cover, which is known to be a (much) more restrictive parameter than treewidth. Leveraging the latter algorithm, [23] also present an FPT approximation scheme which can  $(1 - \varepsilon)$ -approximate the problem in time  $2^{O(vc/\varepsilon)} n^{O(1)}$ , that is, single-exponential time with respect to  $vc$ .

**Our contribution.** We begin our work by considering MAX MIN FVS parameterized by the most standard structural parameter, treewidth. We observe that, using standard DP techniques, we can obtain an algorithm running in time  $tw^{O(tw)} n^{O(1)}$ , that is, slightly super-exponential with respect to treewidth. Note that this slightly super-exponential running time is already present in the  $vc^{O(vc)} n^{O(1)}$  algorithm of [23], despite the fact that vertex cover is a much more severely restricted parameter. Hence, our algorithm generalizes the algorithm of [23] without a significant sacrifice in the running time.

Despite the above, our main contribution with respect to structural parameters is not our algorithm for parameter treewidth, but an answer to a question that is naturally posed given the above: can the super-exponential dependence present in both our algorithm and the algorithm of [23] be avoided, that is, can we obtain a  $2^{O(tw)} n^{O(1)}$  algorithm? We show that this is likely impossible, as the existence of an algorithm running in time  $vc^{o(vc)} n^{O(1)}$  is ruled out by the ETH (and hence also the existence of a  $tw^{o(tw)} n^{O(1)}$  algorithm). This result is likely to be of wider interest to the parameterized complexity community, where one of the most exciting developments of the last fifteen years has arguably been the development of the Cut&Count technique (and its variations). One of the crowning achievements of this

---

<sup>1</sup> Throughout the paper we assume that the reader is familiar with the basics of parameterized complexity, as given in standard textbooks [16].

technique is the design of single-exponential algorithms for connectivity problems – indeed an algorithm running in time  $3^{\text{tw}}n$  for MINIMUM FVS is given in [17]. It has therefore been of much interest to understand which connectivity problems admit single-exponential algorithms using such techniques (see e.g. [7] and the references within). Curiously, even though several cousins of MINIMUM FEEDBACK VERTEX SET have been considered in this context (such as SUBSET FEEDBACK VERTEX SET and RESTRICTED EDGE-SUBSET FEEDBACK EDGE SET), for MAX MIN FVS, which is arguably a very natural variant, it was not known whether a single-exponential algorithm for parameter treewidth is possible. Our work thus adds to the literature a natural connectivity problem where Cut&Count can provably not be applied (under standard assumptions). Interestingly, our lower bound even applies to the case of vertex cover, which is rare, as most problems tend to become rather easy under this very restrictive parameter.

We then move on to consider the parameterization of the problem by  $k$ , the size of the sought solution. Observe that a  $k^{O(k)}n^{O(1)}$  algorithm can easily be obtained by the results sketched above and a simple win/win argument: start with any minimal feedback vertex set  $S$  of the given graph  $G$ : if  $|S| \geq k$  we are done; if not, then  $\text{tw}(G) \leq k$  and we can solve the problem using the algorithm for treewidth. It is therefore only interesting to consider algorithms with a single-exponential dependence on  $k$ . Such an algorithm, with complexity  $10^k n^{O(1)}$ , was claimed by [23]. Unfortunately, as we explain in detail in Section 5, this algorithm contains a significant flaw<sup>2</sup>.

Our contribution is to present a corrected version of the algorithm of [23], which also achieves a slightly better running time of  $9.34^k n^{O(1)}$ , compared to the  $10^k n^{O(1)}$  of the (flawed) algorithm of [23]. Our algorithm follows the same general strategy of [23], branching and placing vertices in the forest or the feedback vertex set. However, we have to rely on a more sophisticated measure of progress, because simply counting the size of the selected set is not sufficient. We therefore measure our progress towards a restricted special case we identify, namely the case where the undecided part of the graph induces a linear forest. Though this special case sounds tantalizingly simple, we show that the problem is still NP-complete under this restriction, but obtaining an FPT algorithm is much easier. We then plug in our algorithm to a more involved branching procedure which aims to either reduce instances into this special case, or output a certifiable minimal feedback vertex set of the desired size.

Finally, motivated by the above we note that a blocking point in the design of algorithms for MAX MIN FVS seems to be the difficulty of the extension problem: given a set  $S_0$ , decide if a minimal fvs  $S$  that extends  $S_0$  exists. As mentioned, Casel et al. [13] showed that this problem is W[1]-hard parameterized by  $|S_0|$ . Intriguingly, however, it is not even known if this problem is in XP, that is, whether it is solvable in polynomial time for fixed  $k$ . We show that this is perhaps not surprising, as obtaining a polynomial time algorithm in this case would imply the existence of a polynomial time algorithm for the notorious  $k$ -IN-A-TREE problem: given  $k$  terminals in a graph, find an induced tree that contains them. Since this problem was solved for  $k = 3$  in a breakthrough by Chudnovsky and Seymour [15], the complexity for fixed  $k \geq 4$  has remained a big open problem (for example [29] states that “Solving it in polynomial time for constant  $k$  would be a huge result”). It is therefore perhaps not surprising that obtaining an XP algorithm for the extension problem for minimal feedback vertex sets of fixed size is challenging, since such an algorithm would settle another long-standing problem.

---

<sup>2</sup> Saket Saurabh, one of the authors of [23], confirmed so via private communication with Michael Lampis.

**Other relevant work.** As mentioned, MAX MIN FVS is an example of a wider class of MaxMin problems which have recently attracted much attention in the literature, among the most well-studied of which are MAXIMUM MINIMAL VERTEX COVER [2, 11, 12, 34] and UPPER DOMINATING SET (which is the standard name for MAXIMUM MINIMAL DOMINATING SET) [1, 3, 5, 21]. Besides these problems, MaxMin or MinMax versions of cut and separations problems [19, 26, 30], knapsack problems [22, 24], matching problems [14], and coloring problems [6] have also been studied.

The question of which connectivity problems admit single-exponential algorithms parameterized by treewidth has been well-studied over the last decade. As mentioned, the main breakthrough was the discovery of the Cut&Count technique [16], which gave randomized  $2^{O(\text{tw})}n^{O(1)}$  algorithms for many such problems, such as STEINER TREE, HAMILTONICITY, CONNECTED DOMINATING SET and others. Follow-up work also provided deterministic algorithms with complexity  $2^{O(\text{tw})}n^{O(1)}$  [8]. It is important to note that the discovery of these techniques was considered a surprise at the time, as the conventional wisdom was that connectivity problems probably require  $\text{tw}^{O(\text{tw})}$  time to be solved [31]. Naturally, the topic was taken up with much excitement, in an attempt to discover the limits of such techniques, including problems for which they cannot work. In this vein, [33] gave a meta-theorem capturing many tractable problems, and also an example problem that cannot be solved in time  $2^{o(\text{tw}^2)}n^{O(1)}$  under the ETH. Several other examples of connectivity problems which require slightly super-exponential time parameterized by treewidth are now known [4, 27], with the most relevant to our work being the feedback vertex set variants studied in [7, 10], as well as the digraph version of the minimum feedback vertex set problem (parameterized by the treewidth of the underlying graph) [9]. The results of our paper seem to confirm the intuition that the Cut&Count technique is rather fragile when applied to feedback vertex set problems, since in many variations or generalizations of this problem, a super-exponential dependence on treewidth is inevitable (assuming the ETH).

## 2 Preliminaries

Throughout the paper, we use standard graph notation [18]. Moreover, for vertex  $u \in V(G)$ , let  $\deg_X(u)$  denote its degree in  $G[X \cup \{u\}]$ , where  $X \subseteq V(G)$ . A *multigraph*  $G$  is a graph which is permitted to have multiple edges with the same end nodes, thus, two vertices may be connected by more than one edge. Given a (multi)graph  $G$ , where  $e = \{u, v\} \in E(G)$  is a not necessarily unique edge connecting distinct vertices  $u$  and  $v$ , the *contraction* of  $e$  results in a new graph  $G'$  such that  $V(G') = (V(G) \setminus \{u, v\}) \cup \{w\}$ , while for each edge  $\{u, x\}$  or  $\{v, x\}$  in  $E(G)$ , there exists an edge  $\{w, x\}$  in  $E(G')$ . Any edge  $e \in E(G)$  not incident to  $u, v$  also belongs to  $E(G')$ . If  $u$  and  $v$  were additionally connected by an edge apart from  $e$ , then  $w$  has a self loop.

For  $i \in \mathbb{N}$ ,  $[i]$  denotes the set  $\{1, \dots, i\}$ . A feedback vertex set  $S$  of  $G$  is minimal if and only if  $\forall s \in S$ ,  $G[(V(G) \setminus S) \cup \{s\}]$  contains a cycle, namely a *private cycle* of  $s$  [21]. Lastly, we make use of a weaker version of ETH, which states that 3-SAT cannot be determined in time  $2^{o(n)}$ , where  $n$  denotes the number of the variables [28].

Finally, note that the proofs of all lemmas and theorems marked with  $(\star)$  are present in the full version of the paper.

## 3 Treewidth Algorithm

Here we will present an algorithm for MAX MIN FVS parameterized by the treewidth of the input graph, arguably the most well studied structural parameter. As a corollary of the lower bound established in Section 4, it follows that the running time of the algorithm is essentially optimal under the ETH.

► **Theorem 1.** ( $\star$ ) *Given an instance  $\mathcal{I} = (G, k)$  of MAX MIN FVS, as well as a nice tree decomposition of  $G$  of width  $tw$ , there exists an algorithm that decides  $\mathcal{I}$  in time  $tw^{O(tw)}n^{O(1)}$ .*

**Proof sketch.** The main idea lies on performing standard dynamic programming on the nodes of the nice tree decomposition. To this end, for each node, we will consider all the partial solutions, corresponding to (not necessarily minimal) feedback vertex sets of the subgraph induced by the vertices of the nodes of the corresponding subtree of the tree decomposition. We will try to extend such a feedback vertex set to a minimal feedback vertex set of  $G$ , that respects the partial solution. For each partial solution, it is imperative to identify, apart from the vertices of the bag that belong to the feedback vertex set, the connectivity of the rest of the vertices in the potential final forest. In order to do so, we consider a coloring indicating that, same colored vertices of the forest of the partial solution, should be in the same connected component of the potential final forest. Moreover, we keep track of which vertices of the forest of the partial solution are connected via paths containing forgotten vertices. Finally, for each vertex of the feedback vertex set of the partial solution, we need to identify one of its private cycles. To do so, we first guess the connected component of the potential final forest that “includes” such a private cycle, while additionally keeping track of the number of edges between the vertex and said component. ◀

#### 4 ETH Lower Bound

In this section we present a lower bound on the complexity of solving MAX MIN FVS parameterized by vertex cover. Starting from a 3-SAT instance on  $n$  variables, we produce an equivalent MAX MIN FVS instance on a graph of vertex cover  $O(n/\log n)$ , hence any algorithm solving the latter problem in time  $vc^{o(vc)}n^{O(1)}$  would refute the ETH. As already mentioned, vertex cover is a very restrictive structural parameter, and due to known relationships of vertex cover with more general parameters, such as treedepth and treewidth, analogous lower bounds follow for these parameters. We first state the main theorem.

► **Theorem 2.** *There is no  $vc^{o(vc)}n^{O(1)}$  time algorithm for MAX MIN FVS, where  $vc$  denotes the size of the minimum vertex cover of the input graph, unless the ETH fails.*

Before we present the details of our construction, let us give some high-level intuition. Our goal is to “compress” an  $n$ -variable instance of 3-SAT, into an MAX MIN FVS instance with vertex cover roughly  $n/\log n$ . To this end, we will construct  $\log n$  choice gadgets, each of which is supposed to represent  $n/\log n$  variables, while contributing only  $n/\log^2 n$  to the vertex cover. Hence, each vertex of each such gadget must be capable of representing roughly  $\log n$  variables.

Our choice gadget may be thought of as a variation of a bipartite graph with sets  $L, R$ , of size roughly  $n/\log^2 n$  and  $\sqrt{n}$  respectively. If one naively tries to encode information in such a gadget by selecting which vertices of  $L \cup R$  belong in an optimal solution, this would only give 2 choices per vertex, which is not efficient enough. Instead, we engineer things in a way that all vertices of  $L \cup R$  must belong in the forest in an optimal solution, and the interesting choice for a vertex  $\ell$  of  $L$  is with which vertex  $r$  of  $R$  we will place  $\ell$  in the same component. In this sense, a vertex  $\ell$  of  $L$  has  $|R|$  choices, which is sufficient to encode the assignment for  $\Omega(\log n)$  variables. What remains, then, is to add machinery that enforces this basic setup, and then clause checking vertices which for each clause verify that the clause is satisfied by testing if an  $\ell$  vertex that represents one of its literals is in the same component as an  $r$  vertex that represents a satisfying assignment for the clause.

### 4.1 Preliminary Tools

Before we present the construction that proves Theorem 2, we give a variant of 3-SAT from which it will be more convenient to start our reduction, as well as a basic force gadget that we will use in our construction to ensure that some vertices must be placed in the forest in order to achieve an optimal solution.

**3P3SAT.** We first define a constrained version of 3-SAT, called 3-PARTITIONED-3-SAT (3P3SAT for short), and establish its hardness under the ETH.

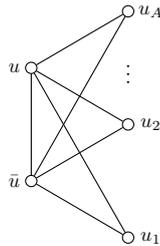
**3-PARTITIONED-3-SAT**

**Input:** A formula  $\phi$  in 3-CNF form, together with a partition of the set of its variables  $V$  into three disjoint sets  $V_1, V_2, V_3$ , with  $|V_i| = n$ , such that no clause contains more than one variable from each  $V_i$ .

**Task:** Determine whether  $\phi$  is satisfiable.

► **Theorem 3.** (★) *3-PARTITIONED-3-SAT cannot be decided in time  $2^{o(n)}$ , unless the ETH fails.*

**Force gadgets.** We now present a gadget that will ensure that a vertex  $u$  must be placed in the forest in any solution that finds a large minimal feedback vertex set. In the remainder, suppose that  $A$  is a sufficiently large value (we give a concrete value to  $A$  in the next section). When we say that we attach a *force gadget* to a vertex  $u$ , we introduce  $A + 1$  new vertices  $\bar{u}, u_1, \dots, u_A$  to the graph such that the vertices  $u_i$  form an independent set, while there exist edges  $\{u, u_i\}, \{\bar{u}, u_i\}$  for all  $i \in [A]$ , as well as the edge  $\{u, \bar{u}\}$ . We refer to vertex  $\bar{u}$  as the *gadget twin* of  $u$ , while the rest of the vertices will be referred to as the *gadget leaves* of  $u$ . Intuitively, the idea here is that if  $u$  (or  $\bar{u}$ ) is contained in a minimal feedback vertex set, then none of the  $A$  leaves of the gadget can be taken, because these vertices cannot have private cycles. Hence, setting  $A$  to be sufficiently large will allow us to force  $u$  to be in the forest.



■ **Figure 1** Force gadget attached to vertex  $u$ .

### 4.2 Construction

Let  $\phi$  be a 3P3SAT instance of  $m$  clauses, where  $|V_p| = n$  for  $p \in [3]$  and, without loss of generality, assume that  $n$  is a power of 4 (this can be achieved by adding dummy variables to the instance if needed). Partition each variable set  $V_p$  to  $\log n$  subsets  $V_p^q$  of size at most  $\lceil \frac{n}{\log n} \rceil$ , where  $p \in [3]$  and  $q \in [\log n]$ . Let  $L = \lceil \frac{n}{\log^2 n} \rceil$ . Moreover, partition each variable subset  $V_p^q$  into  $2L$  subsets  $\mathcal{V}_\alpha^{p,q}$  of size as equal as possible, where  $\alpha \in [2L]$ . In the following

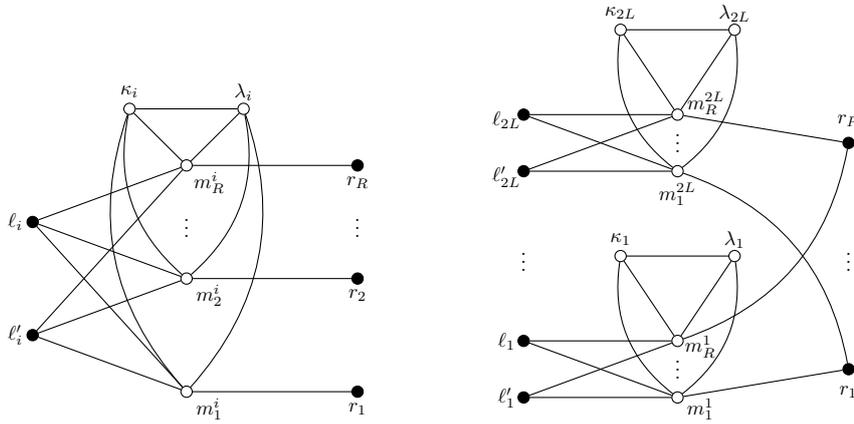
we will omit  $p$  and  $q$  and instead use the notation  $\mathcal{V}_\alpha$ , whenever  $p, q$  are clear from the context. Define  $R = \sqrt{n}$ ,  $A = n^2 + m$  and  $k = (4AL + AR + 2LR) \cdot 3 \log n + m$ . We will proceed with the construction of a graph  $G$  such that  $G$  has a minimal feedback vertex set of size at least  $k$  if and only if  $\phi$  is satisfiable.

For each variable subset  $V_p^q$ , we define the choice gadget graph  $G_p^q$  as follows:

- $V(G_p^q) = \{\ell_i, \ell'_i, \kappa_i, \lambda_i \mid i \in [2L]\} \cup \{r_j \mid j \in [R]\} \cup \{m_j^i \mid i \in [2L], j \in [R]\}$ ,
- all the vertices  $\ell_i, \ell'_i$  and  $r_j$  have an attached force gadget,
- for  $i \in [2L]$ ,  $N(\kappa_i) = M_i \cup \{\lambda_i\}$  and  $N(\lambda_i) = M_i \cup \{\kappa_i\}$ , where  $M_i = \{m_j^i \mid j \in [R]\}$ ,
- for  $i \in [2L]$  and  $j \in [R]$ ,  $m_j^i$  has an edge with  $\ell_i, \ell'_i$  and  $r_j$ .

We will refer to the set  $X_i = M_i \cup \{\kappa_i, \lambda_i\}$  as the *choice set*  $i$ .

Intuitively, one can think of this gadget as having been constructed as follows: we start with a complete bipartite graph that has on one side the vertices  $\ell_i$  and on the other the vertices  $r_j$ ; we subdivide each edge of this graph, giving the vertices  $m_j^i$ ; for each  $i \in [2L]$  we add  $\ell'_i, \kappa_i, \lambda_i$ , connect them to the same  $m_j^i$  vertices that  $\ell_i$  is connected to and connect  $\kappa_i$  to  $\lambda_i$ ; we attach force gadgets to all  $\ell_i, \ell'_i, r_j$ . Hence, as sketched before, the idea of this gadget is that the choice of a vertex  $\ell_i$  is to pick an  $r_j$  with which it will be in the same component in the forest, and this will be expressed by picking one  $m_j^i$  that will be placed in the forest.



(a) Part of the construction concerning  $X_i$ . (b) The whole choice gadget graph  $G_p^q$ .

■ **Figure 2** Black vertices have a force gadget attached.

Each vertex  $\ell_\alpha$  of  $G_p^q$  is used to represent a variable subset  $\mathcal{V}_\alpha^{p,q} \subseteq V_p^q$  containing at most

$$|\mathcal{V}_\alpha^{p,q}| \leq \left\lceil \frac{|V^{p,q}|}{2L} \right\rceil \leq \left\lceil \frac{\lceil \frac{n}{\log n} \rceil}{2L} \right\rceil = \left\lceil \frac{n}{2L \log n} \right\rceil \leq \left\lceil \frac{n}{2 \frac{n}{\log^2 n} \log n} \right\rceil = \left\lceil \frac{\log n}{2} \right\rceil = \frac{\log n}{2}$$

variables of  $\phi$ , where we used Theorem 3.10 of [25], for  $f(x) = x/2L$ . We fix an arbitrary one-to-one mapping so that every vertex  $m_\beta^\alpha$ , where  $\beta \in [R]$ , corresponds to a different assignment for this subset, which is dictated by which element of  $M_\alpha$  was not included in the final feedback vertex set. Since  $R = 2^{\log n/2} = \sqrt{n}$ , the size of  $M_\alpha$  is sufficient to uniquely encode all the different assignments of  $\mathcal{V}_\alpha$ .

Finally, introduce vertices  $c_i$ , where  $i \in [m]$ , each of which corresponds to a clause of  $\phi$ , and define graph  $G$  as the union of these vertices as well as all graphs  $G_p^q$ , where  $p \in [3]$  and  $q \in [\log n]$ . For a clause vertex  $c$ , add an edge to  $\ell_\alpha$  when  $\mathcal{V}_\alpha$  contains a variable appearing in  $c$ , as well as to the vertices  $r_\beta$  for each such  $\ell_\alpha$ , such that  $m_\beta^\alpha \notin S$  corresponds to an assignment of  $\mathcal{V}_\alpha$  satisfying  $c$ , where  $S$  denotes a minimal feedback vertex set. Notice

that since no clause contains multiple variables from the same variable set  $V_i$ , due to the refinement of the partition of the variables, it holds that all the variables of a clause will be represented by vertices appearing in distinct  $G_p^q$ .

### 4.3 Correctness

Having constructed the previously described instance  $(G, k)$  of MAX MIN FVS, it remains to prove its equivalence with the initial 3-PARTITIONED-3-SAT instance.

► **Lemma 4.** ( $\star$ ) *Any minimal feedback vertex set  $S$  of  $G$  of size at least  $k$  has the following properties:*

- (i)  $S$  does not contain any vertex attached with a force gadget or its gadget twin,
  - (ii)  $|M_i \setminus S| \leq 1$ , for every  $G_p^q$  and  $i \in [2L]$ ,
  - (iii)  $|S \cap V(G_p^q)| = 4AL + AR + 2LR$ ,
- where  $p \in [3]$  and  $q \in [\log n]$ .

► **Lemma 5.** ( $\star$ ) *If  $\phi$  has a satisfying assignment, then  $G$  has a minimal feedback vertex set of size at least  $k$ .*

► **Lemma 6.** ( $\star$ ) *If  $G$  has a minimal feedback vertex set of size at least  $k$ , then  $\phi$  has a satisfying assignment.*

► **Lemma 7.** ( $\star$ ) *It holds that  $\text{vc}(G) = O(n/\log n)$ .*

Using the previous lemmas, we can prove Theorem 2.

**Proof of Theorem 2.** Let  $\phi$  be a 3-PARTITIONED-3-SAT formula. In polynomial time, we can construct a graph  $G$  such that, due to Lemmas 5 and 6, deciding if  $G$  has a minimal feedback vertex set of size at least  $k$  is equivalent to deciding if  $\phi$  has a satisfying assignment. In that case, assuming there exists a  $\text{vc}^{o(\text{vc})}$  algorithm for MAX MIN FVS, one could decide 3-PARTITIONED-3-SAT in time

$$\text{vc}^{o(\text{vc})} = \left( \frac{n}{\log n} \right)^{o(n/\log n)} = 2^{(\log n - \log \log n)o(n/\log n)} = 2^{o(n)},$$

which contradicts the ETH due to Theorem 3. ◀

Since for any graph  $G$  it holds that  $\text{tw}(G) \leq \text{vc}(G)$ , the following corollary holds.

► **Corollary 8.** *There is no  $\text{tw}^{o(\text{tw})}n^{O(1)}$  time algorithm for MAX MIN FVS, where  $\text{tw}$  denotes the treewidth of the input graph, unless the ETH fails.*

## 5 Natural Parameter Algorithm

In this section we will present an FPT algorithm for MAX MIN FVS parameterized by the natural parameter, i.e. the size of the maximum minimal feedback vertex set  $k$ . The main theorem of this section is the following.

► **Theorem 9.** *MAX MIN FVS can be solved in time  $9.34^k n^{O(1)}$ .*

**Structure of the Section.** In Section 5.1 we define the closely related ANNOTATED MMFVS problem, and prove that it remains NP-hard, even on some instances of specific form, called *path restricted instances*. Subsequently, we present an algorithm dealing with this kind of instances, which either returns a minimal feedback vertex set of size at least  $k$  or concludes that this is a No instance of ANNOTATED MMFVS. Afterwards, in Section 5.2, we solve MAX MIN FVS by producing a number of instances of ANNOTATED MMFVS and utilizing the previous algorithm, therefore proving Theorem 9.

**Oversight of [23].** The algorithm of [23] performs a branching procedure which marks vertices as either belonging in the feedback vertex set or the remaining forest. The flaw is that the algorithm ceases the branching once  $k$  vertices have been identified as vertices of the feedback vertex set. However, this is not correct, since deciding if a given set  $S_0$  can be extended into a minimal feedback vertex set  $S \supseteq S_0$  is NP-complete and even W[1]-hard parameterized by  $|S_0|$  [13]. Hence, identifying  $k$  vertices of the solution is not, in general, sufficient to produce a feasible solution and the algorithm of [23] is incomplete, because it does not explain how the guessed part of the feedback vertex set can be extended into a feasible minimal solution. Intuitively, the pitfall here is that, unlike other standard maximization problems, such as MAX CLIQUE, MAX MIN FVS is not monotone, that is, a graph that contains a feasible solution of size  $k$  is not guaranteed to contain a feasible solution of size  $k - 1$  (consider, for instance, a  $K_{2,n}$ ).

## 5.1 Annotated MMFVS and Path Restricted Instances

First, we define the following closely related problem, denoted by ANNOTATED MMFVS for short.

ANNOTATED MAXIMUM MINIMAL FEEDBACK VERTEX SET

**Input:** A graph  $G = (V, E)$ , disjoint sets  $S, F \subseteq V$  where  $S \cup F$  is a feedback vertex set of  $G$ , as well as an integer  $k$ .

**Task:** Determine whether there exists a minimal feedback vertex set  $S'$  of  $G$  of size  $|S'| \geq k$  such that  $S' \supseteq S$  and  $S' \cap F = \emptyset$ .

**Remarks.** Notice that if  $F$  is not a forest, then the corresponding instance always has a negative answer. For the rest of this section, let  $U = V(G) \setminus (S \cup F)$ . Moreover, let  $H = \{s \in S \mid \deg_F(s) \geq 2 \text{ and } \deg_U(s) \leq 1\}$  denote the set of *good vertices* of  $S$ . An *interesting path* of  $G[U]$  is a connected component of  $G[U]$  such that for every vertex  $u$  belonging to said component, it holds that  $\deg_{F \cup U}(u) = 2$ . If every connected component of  $G[U]$  is an interesting path, then this is a *path restricted instance*. Furthermore, given an instance  $\mathcal{I}$ , let  $\text{ammfvs}(\mathcal{I})$  be equal to 1 if it is a Yes instance and 0 otherwise.

Let  $\mathcal{I} = (G, S, F, k)$  be a path restricted instance of ANNOTATED MMFVS. We will present an algorithm that either returns a minimal feedback vertex set  $S' \subseteq S \cup U$  of  $G$  of size at least  $k$  or concludes that this is a No instance of ANNOTATED MMFVS. Notice that ANNOTATED MMFVS remains NP-hard even on such instances, as dictated by Theorem 10. Therefore, we should not expect to solve path restricted instances of ANNOTATED MMFVS in polynomial time.

► **Theorem 10.** ( $\star$ ) ANNOTATED MMFVS is NP-hard on path restricted instances, even if all the paths are of length 2.

## 62:10 Parameterized Max Min Feedback Vertex Set

We proceed by presenting the main algorithm of this subsection, which will be essential in proving Theorem 9.

► **Theorem 11.** ( $\star$ ) *Let  $\mathcal{I} = (G, S, F, k)$  be a path restricted instance of ANNOTATED MMFVS, and let  $g$  denote the number of its good vertices. There is an algorithm running in time  $O(3^{k-g}n^{O(1)})$  which either returns a minimal feedback vertex set  $S' \subseteq S \cup U$  of  $G$  of size at least  $k$  or concludes that  $\mathcal{I}$  is a No instance of ANNOTATED MMFVS.*

**Proof sketch.** The main idea of the algorithm lies on the fact that we can efficiently handle instances where either  $k = 0$  or  $S = \emptyset$ . Towards this, we will employ a branching strategy that, as long as  $S$  remains non empty, new instances with reduced  $k$  are produced. Prior to performing branching, we first observe that we can efficiently deal with the good vertices. Afterwards, by employing said branching strategy, in every step we decide which vertex will be counted towards the  $k$  required, thereby reducing parameter  $k$  on each iteration. If at some point  $k = 0$  or  $S = \emptyset$ , it remains to decide whether this comprises a viable solution  $S'$ . Notice that  $S'$  may not be a solution for the annotated instance, since even if  $|S'| \geq k$ , it does not necessarily hold that  $S' \supseteq S$ . ◀

### 5.2 Algorithm for Max Min FVS

We start by presenting a high level sketch of the algorithm for MAX MIN FVS. The starting point is a minimal feedback vertex set  $S_0$  of  $G$ . Note that such a set can be obtained in polynomial time, while if it is of size at least  $k$ , we are done. Therefore, assume that  $|S_0| < k$ . Then, assuming there exists a minimal feedback vertex set  $S^*$ , where  $|S^*| \geq k$  and  $F^* = V(G) \setminus S^*$ , we will guess  $S_0 \cap S^*$ , thereby producing instances  $\mathcal{I}_0 = (G, S_0 \cap S^*, S_0 \cap F^*, k)$  of ANNOTATED MMFVS. Subsequently, we will establish a number of *safe* reduction rules, which do not affect the answer of the instances. We will present a measure of progress  $\mu$ , which guarantees that if an instance  $\mathcal{I} = (G, S, F, k)$  of ANNOTATED MMFVS has  $\mu(\mathcal{I}) \leq 1$ , then  $G$  has a minimal feedback vertex set  $S' \subseteq S \cup U$  of size at least  $k$ . Then, we will employ a branching strategy which, given  $\mathcal{I}_i$ , will produce instances  $\mathcal{I}_{i+1}^1, \mathcal{I}_{i+1}^2$  of lesser measure of progress, such that  $\mathcal{I}_i$  is a Yes instance if and only if at least one of  $\mathcal{I}_{i+1}^1, \mathcal{I}_{i+1}^2$  is also a Yes instance. If we can no further apply our branching strategy, and the measure of progress remains greater than 1, then it holds that  $\mathcal{I}$  is a path restricted instance and Theorem 11 applies.

**Measure of progress.** Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS. We define as  $\mu(\mathcal{I}) = k + cc(F) - g - p$  its *measure of progress*, where

- $cc(F)$  denotes the number of connected components of  $F$ ,
- $g$  denotes the number of good vertices of  $S$ ,
- $p$  denotes the number of interesting paths of  $G[U]$ .

It holds that if  $\mu(\mathcal{I}) \leq 1$ , then the underlying MAX MIN FVS instance has a positive answer, which does not necessarily respect the constraints dictated by the annotated version.

► **Lemma 12.** ( $\star$ ) *Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS, where  $\mu(\mathcal{I}) \leq 1$ . Then,  $G$  has a minimal feedback vertex set  $S' \subseteq S \cup U$  of size at least  $k$ .*

**Reduction rules.** In the following, we will describe some reduction rules which do not affect the answer of an instance of ANNOTATED MMFVS, while not increasing its measure of progress.

► **Lemma 13.** ( $\star$ ) *Let  $G = (V, E)$  be a (multi)graph and  $uv \in E(G)$ . Then,  $G$  is acyclic if and only if  $G/uv$  is acyclic.*

**Rule 1.** Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS,  $u, v \in F$  and  $uv \in E$ . Then, replace  $\mathcal{I}$  with  $\mathcal{I}' = (G', S, F', k)$ , where  $G' = G/uv$  occurs from the contraction of  $u$  and  $v$  into  $w$ , while  $F' = (F \cup \{w\}) \setminus \{u, v\}$ .

**Rule 2.** Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS,  $u \in U$  and  $\deg_{F \cup U}(u) = 0$ . Then, replace  $\mathcal{I}$  with  $\mathcal{I}' = (G - u, S, F, k)$ .

**Rule 3.** Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS,  $u \in U$  and  $\deg_{F \cup U}(u) = 1$ , while  $v \in N(u) \cap (F \cup U)$ . Then, replace  $\mathcal{I}$  with  $\mathcal{I}' = (G', S, F', k)$ , where  $G' = G/uv$  occurs from the contraction of  $u$  and  $v$  into  $w$ , while  $F' = (F \cup \{w\}) \setminus \{v\}$  if  $v \in F$ , and  $F' = F$  otherwise.

► **Lemma 14.** ( $\star$ ) *Applying rules 1, 2 and 3 does not change the outcome of the algorithm and does not increase the measure of progress.*

After exhaustively applying the aforementioned rules, it holds that  $\forall u \in U$ ,  $\deg_{F \cup U}(u) \geq 2$ , i.e.  $G[U]$  is a forest containing trees, all the leaves of which have at least one edge to  $F$ . Moreover,  $G[F]$  comprises an independent set. We proceed with a branching strategy that produces instances of ANNOTATED MMFVS of reduced measure of progress. If at some point  $\mu \leq 1$ , then Lemma 12 can be applied.

**Branching strategy.** Let  $\mathcal{I} = (G, S, F, k)$  be an instance of ANNOTATED MMFVS, on which all of the reduction rules have been applied exhaustively, thus, it holds that a)  $\forall u \in U$ ,  $\deg_{F \cup U}(u) \geq 2$  and b)  $F$  is an independent set.

Define  $u \in U$  to be an *interesting* vertex if  $\deg_{F \cup U}(u) \geq 3$ . As already noted,  $G[U]$  is a forest, the leaves of which all have an edge towards  $F$ , otherwise Rule 3 could still be applied. Consider a root for each tree of  $G[U]$ . For some tree  $T$ , let  $v$  be an interesting vertex at maximum distance from the corresponding root, i.e.  $v$  is an interesting vertex of maximum height. Notice that such a tree cannot be an interesting path. We branch depending on whether  $u$  is in the feedback vertex set or not. Towards this end, let  $S' = S \cup \{v\}$  and  $F' = F \cup \{v\}$ , while  $\mathcal{I}_1 = (G, S', F, k)$  and  $\mathcal{I}_2 = (G, S, F', k)$ . It holds that  $\mathcal{I}$  is a Yes instance if and only if at least one of  $\mathcal{I}_1, \mathcal{I}_2$  is a Yes instance, while if  $G[F']$  contains a cycle,  $\mathcal{I}_2$  is a No instance and we discard it. We replace  $\mathcal{I}$  with the instances  $\mathcal{I}_1$  and  $\mathcal{I}_2$ .

► **Lemma 15.** ( $\star$ ) *The branching strategy produces instances of reduced measure of progress, without reducing the number of good vertices. Additionally, whenever the branching places a vertex on the feedback vertex set, this vertex is good.*

**Complexity.** Starting from an instance  $(G, k)$  of MAX MIN FVS, we produce a minimal feedback vertex set  $S_0$  of  $G$  in polynomial time. If  $|S_0| \geq k$ , we are done. Alternatively, we produce instances of ANNOTATED MMFVS by guessing the intersection of  $S_0$  with some minimal feedback vertex set of  $G$  of size at least  $k$ . Let  $\mathcal{I} = (G, S, F, k)$  be one such instance. It holds that  $\mu(\mathcal{I}) \leq k + c$ , where  $c = cc(F)$ , therefore the branching will perform at most  $k + c$  steps. Notice that, at any step of the branching procedure, the number of good vertices never decreases. Now, consider a path restricted instance  $\mathcal{I}' = (G', S', F', k)$  resulting from branching starting on  $\mathcal{I}$ , on which branching, exactly  $\ell$  times a vertex was placed in the

## 62:12 Parameterized Max Min Feedback Vertex Set

feedback vertex set, therefore  $|S'| - |S| = \ell$ . There are at most  $\binom{k+c}{\ell}$  different such instances, each of which has at least  $\ell$  good vertices, thus Theorem 11 requires time at most  $3^{k-\ell}n^{O(1)}$ . Since  $0 \leq \ell \leq k+c$ , and there are at most  $\binom{k}{c}$  different instances  $\mathcal{I}$ , the algorithm runs in time  $9.34^k n^{O(1)}$ , since

$$\begin{aligned} \sum_{c=0}^k \binom{k}{c} \sum_{\ell=0}^{k+c} \binom{k+c}{\ell} 3^{k-\ell} &= 3^k \sum_{c=0}^k \binom{k}{c} \sum_{\ell=0}^{k+c} \binom{k+c}{\ell} 3^{-\ell} = 3^k \sum_{c=0}^k \binom{k}{c} \left(\frac{4}{3}\right)^{k+c} \\ &= 4^k \sum_{c=0}^k \binom{k}{c} \left(\frac{4}{3}\right)^c = 4^k \left(\frac{7}{3}\right)^k \leq 9.34^k. \end{aligned}$$

### 6 The Extension Problem

In this section we consider the following extension problem:

MINIMAL FVS EXTENSION

**Input:** A graph  $G = (V, E)$  and a set  $S \subseteq V$ .

**Task:** Determine whether there exists  $S^* \supseteq S$  such that  $S^*$  is a minimal feedback vertex set of  $G$ .

Observe that this is a special case of ANNOTATED MMFVS, since we essentially set  $F = \emptyset$  and do not care about the size of the produced solution, albeit with the difference that now we will not focus on the case where  $V \setminus S$  is already acyclic. This extension problem was already shown to be W[1]-hard parameterized by  $|S|$  by Casel et al. [13]. One question that was left open, however, was whether it is solvable in polynomial time for fixed  $|S|$ , that is, whether it belongs in the class XP. Superficially, this seems somewhat surprising, because for the closely related MAXIMUM MINIMAL VERTEX COVER and UPPER DOMINATING SET problems, membership of the extension problem in XP is almost trivial: it suffices to guess for each  $v \in S$  a private edge or vertex that is only dominated by  $v$ , remove from consideration other vertices that dominate this private edge or vertex, and then attempt to find any feasible solution. The reason that this strategy does not seem to work for feedback vertex set is that for each  $v \in S$  we would have to guess a private cycle. Since a priori we have no bound on the length of such a cycle, there is no obvious way to achieve this task in  $n^{f(k)}$  time.

Though we do not settle the complexity of the extension problem for fixed  $k$ , we provide evidence that obtaining a polynomial time algorithm would be a challenging task, because it would imply a similar algorithm for the  $k$ -IN-A-TREE problem. In the latter, we are given a graph  $G$  and a set  $T$  of  $k$  terminals and are asked to find a set  $T^*$  such that  $T \subseteq T^*$  and  $G[T^*]$  is a tree [15, 29].

► **Theorem 16.**  *$k$ -IN-A-TREE parameterized by  $k$  is fpt-reducible to MINIMAL FVS EXTENSION parameterized by the size of the given set.*

**Proof.** Consider an instance  $G = (V, E)$  of  $k$ -IN-A-TREE, with terminal set  $T$ . Let  $T = \{t_1, \dots, t_k\}$ . We add to the graph  $k-1$  new vertices,  $s_1, \dots, s_{k-1}$  and connect each  $s_i$  to  $t_i$  and to  $t_{i+1}$ , for  $i \in [k-1]$ . We set  $S = \{s_1, \dots, s_{k-1}\}$ . This completes the construction. Clearly, this reduction preserves the value of the parameter.

To see correctness, suppose first that a tree  $T^* \supseteq T$  exists in  $G$ . We set  $S_1 = S \cup (V \setminus T^*)$  in the new graph.  $S_1$  is a feedback vertex set, because removing it from the graph leaves  $T^*$ , which is a tree.  $S_1$  contains  $S$ . Furthermore, if  $S_1$  is not minimal, we greedily remove from it arbitrary vertices until we obtain a minimal feedback vertex set  $S_2$ . We claim that  $S_2$  must

still contain  $S$ . Indeed, each vertex  $s_i$ , for  $i \in [k - 1]$  has a private cycle, since its neighbors  $t_i, t_{i+1} \in T^*$ . For the converse direction, if there exists in the new graph a minimal feedback vertex set  $S^*$  that contains  $S$ , then the remaining forest  $F^* = V \setminus S^*$  must contain  $T$ , since each vertex of  $S$  must have a private cycle in the forest, and vertices of  $S$  have degree 2. Furthermore, all vertices of  $T$  must be in the same component of  $F^*$ , because to obtain a private cycle for  $s_i$ , we must have a path from  $t_i$  to  $t_{i+1}$  in  $F^*$ , for all  $i \in [k - 1]$ . Therefore, in this case we have found an induced tree in  $G$  that contains all terminals. ◀

## 7 Conclusions and Open Problems

We have precisely determined the complexity of MAX MIN FVS with respect to structural parameters from vertex cover to treewidth as being slightly super-exponential. One natural question to consider would then be to examine if the same complexity can be achieved when the problem is parameterized by clique-width. Regarding the complexity of the extension problem for sets of fixed size  $k$ , we have shown that this is at least as hard as the well-known (and wide open)  $k$ -IN-A-TREE problem. Barring a full resolution of this question, it would also be interesting to ask if the converse reduction also holds, which would prove that the two problems are actually equivalent.

---

### References

- 1 Hassan AbouEisha, Shahid Hussain, Vadim V. Lozin, Jérôme Monnot, Bernard Ries, and Viktor Zamaraev. Upper domination: Towards a dichotomy through boundary properties. *Algorithmica*, 80(10):2799–2817, 2018. doi:10.1007/s00453-017-0346-9.
- 2 Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. Introducing lop-kernels: A framework for kernelization lower bounds. *Algorithmica*, 84(11):3365–3406, 2022. doi:10.1007/s00453-022-00979-z.
- 3 Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. Parameterized complexity of computing maximum minimal blocking and hitting sets. *Algorithmica*, 85(2):444–491, 2023. doi:10.1007/s00453-022-01036-5.
- 4 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *SODA*, pages 951–970. SIAM, 2020. doi:10.1137/1.9781611975994.57.
- 5 Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th. Paschos. The many facets of upper domination. *Theor. Comput. Sci.*, 717:2–25, 2018. doi:10.1016/j.tcs.2017.05.042.
- 6 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. *SIAM Journal on Discrete Mathematics*, 36(3):1761–1787, 2022. doi:10.1137/20M1385779.
- 7 Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close relatives of feedback vertex set without single-exponential algorithms parameterized by treewidth. In *IPEC*, volume 180 of *LIPICs*, pages 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.3.
- 8 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
- 9 Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In *WG*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018. doi:10.1007/978-3-030-00256-5\_6.

- 10 Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: Chordality is the key to single-exponential parameterized algorithms. *Algorithmica*, 81(10):3890–3935, 2019. doi:10.1007/s00453-019-00579-4.
- 11 Édouard Bonnet, Michael Lampis, and Vangelis Th. Paschos. Time-approximation trade-offs for inapproximable problems. *J. Comput. Syst. Sci.*, 92:171–180, 2018. doi:10.1016/j.jcss.2017.09.009.
- 12 Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. *Discret. Appl. Math.*, 196:62–71, 2015. doi:10.1016/j.dam.2014.06.001.
- 13 Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. On the complexity of solution extension of optimization problems. *Theor. Comput. Sci.*, 904:48–65, 2022. doi:10.1016/j.tcs.2021.10.017.
- 14 Juhi Chaudhary, Sounaka Mishra, and B. S. Panda. Minimum maximal acyclic matching in proper interval graphs. In *CALDAM*, volume 13947 of *Lecture Notes in Computer Science*, pages 377–388. Springer, 2023. doi:10.1007/978-3-031-25211-2\_29.
- 15 Maria Chudnovsky and Paul D. Seymour. The three-in-a-tree problem. *Comb.*, 30(4):387–417, 2010. doi:10.1007/s00493-010-2334-4.
- 16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 17 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.
- 18 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.
- 19 Gabriel L. Duarte, Hiroshi Eto, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Daniel Lokshtanov, Lehilton L. C. Pedrosa, Rafael C. S. Schouery, and Uéverton S. Souza. Computing the largest bond and the maximum connected cut of a graph. *Algorithmica*, 83(5):1421–1458, 2021. doi:10.1007/s00453-020-00789-1.
- 20 Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (in)approximability of maximum minimal FVS. *J. Comput. Syst. Sci.*, 124:26–40, 2022. doi:10.1016/j.jcss.2021.09.001.
- 21 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. *Theor. Comput. Sci.*, 923:271–291, 2022. doi:10.1016/j.tcs.2022.05.013.
- 22 Fabio Furini, Ivana Ljubic, and Markus Sinnl. An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. *Eur. J. Oper. Res.*, 262(2):438–448, 2017. doi:10.1016/j.ejor.2017.03.061.
- 23 Ajinkya Gaikwad, Hitendra Kumar, Soumen Maity, Saket Saurabh, and Shuvam Kant Tripathi. Maximum minimal feedback vertex set: A parameterized perspective. *CoRR*, abs/2208.01953, 2022. doi:10.48550/arXiv.2208.01953.
- 24 Laurent Gourvès, Jérôme Monnot, and Aris Pagourtzis. The lazy bureaucrat problem with common arrivals and deadlines: Approximation and mechanism design. In *FCT*, volume 8070 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2013. doi:10.1007/978-3-642-40164-0\_18.
- 25 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science, 2nd Ed.* Addison-Wesley, 1994.
- 26 Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, and Tsuyoshi Yagita. Finding a maximum minimal separator: Graph classes and fixed-parameter tractability. *Theor. Comput. Sci.*, 865:131–140, 2021. doi:10.1016/j.tcs.2021.03.006.
- 27 Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Digraph coloring and distance to acyclicity. In *STACS*, volume 187 of *LIPICs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.41.

- 28 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.
- 29 Kai-Yuan Lai, Hsueh-I Lu, and Mikkel Thorup. Three-in-a-tree in near linear time. In *STOC*, pages 1279–1292. ACM, 2020. doi:10.1145/3357713.3384235.
- 30 Michael Lampis. Minimum stable cut and treewidth. In *ICALP*, volume 198 of *LIPICs*, pages 92:1–92:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.92.
- 31 Daniel Lokshantov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.
- 32 Sounaka Mishra and Kripasindhu Sikdar. On the hardness of approximating some np-optimization problems related to minimum linear ordering problem. *RAIRO Theor. Informatics Appl.*, 35(3):287–309, 2001. doi:10.1051/ita:2001121.
- 33 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0\_47.
- 34 Meirav Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM J. Discret. Math.*, 31(4):2440–2456, 2017. doi:10.1137/16M109017X.



# Distributed Merlin-Arthur Synthesis of Quantum States and Its Applications

François Le Gall ✉

Graduate School of Mathematics, Nagoya University, Japan

Masayuki Miyamoto ✉

Graduate School of Mathematics, Nagoya University, Japan

Harumichi Nishimura ✉

Graduate School of Informatics, Nagoya University, Japan

---

## Abstract

The generation and verification of quantum states are fundamental tasks for quantum information processing that have recently been investigated by Irani, Natarajan, Nirkhe, Rao and Yuen [CCC 2022], Rosenthal and Yuen [ITCS 2022], Metger and Yuen [QIP 2023] under the term *state synthesis*. This paper studies this concept from the viewpoint of quantum distributed computing, and especially distributed quantum Merlin-Arthur (dQMA) protocols. We first introduce a novel task, on a line, called state generation with distributed inputs (SGDI). In this task, the goal is to generate the quantum state  $U|\psi\rangle$  at the rightmost node of the line, where  $|\psi\rangle$  is a quantum state given at the leftmost node and  $U$  is a unitary matrix whose description is distributed over the nodes of the line. We give a dQMA protocol for SGDI and utilize this protocol to construct a dQMA protocol for the Set Equality problem studied by Naor, Parter and Yegorov [SODA 2020], and complement our protocol by showing classical lower bounds for this problem. Our second contribution is a dQMA protocol, based on a recent work by Zhu and Hayashi [Physical Review A, 2019], to create EPR-pairs between adjacent nodes of a network without quantum communication. As an application of this dQMA protocol, we prove a general result showing how to convert any dQMA protocol on an arbitrary network into another dQMA protocol where the verification stage does not require any quantum communication.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms; Theory of computation → Quantum computation theory

**Keywords and phrases** distributed quantum Merlin-Arthur, distributed verification, quantum computation

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.63

**Related Version** *Full Version*: <https://arxiv.org/abs/2210.01389>

**Funding** FLG was supported by the JSPS KAKENHI grants JP16H01705, JP19H04066, JP20H00579, JP20H04139, JP20H05966, JP21H04879 and by the MEXT Q-LEAP grants JPMXS0118067394 and JPMXS0120319794. MM would like to take this opportunity to thank the “Nagoya University Interdisciplinary Frontier Fellowship” supported by Nagoya University and JST, the establishment of university fellowships towards the creation of science technology innovation, Grant Number JP-MJFS212. HN was supported by the JSPS KAKENHI grants JP19H04066, JP20H05966, JP21H04879, JP22H00522 and by the MEXT Q-LEAP grants JPMXS0120319794.

## 1 Introduction

While quantum computational complexity has so far mostly investigated the complexity of classical problems (e.g., computing Boolean functions) in the quantum setting, recent works [1, 16, 20, 24, 29, 35] have started investigating the complexity of *quantum* problems (e.g., generating quantum states). For instance, Ji, Liu and Song [20] and Kretschmer [24]



© François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 63; pp. 63:1–63:15

Leibniz International Proceedings in Informatics



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

have investigated the concept of quantum pseudorandom states from complexity-theoretic and cryptographic perspectives. Irani, Natarajan, Nirkhe, Rao, and Yuen [16] have made in-depth investigations of the complexity of the *state synthesis problem* in a setting first introduced by Aaronson [1] where the goal is to generate a quantum state by making queries to a classical oracle encoding the state. Rosenthal and Yuen [35] and Metger and Yuen [29] have considered interactive proofs for synthesizing quantum states (and also for implementing unitaries). Here the main goal is to generate complicated quantum states (e.g., quantum states described by an exponential-size generating quantum circuit) efficiently with the help of an all-powerful but untrusted prover. Note that in settings where an all-powerful prover is present, the task of quantum state synthesis is closely related to the task of quantum state verification (since the prover can simply send the quantum state that needs to be synthesized).

In this paper, we investigate the task of state generation and verification in the setting of quantum distributed computing. Quantum distributed computing is a fairly recent research topic: despite early investigations in the 2000s and the 2010s [3, 8, 9, 13, 36], it is only in the past five years that significant advances have been done in understanding the power of quantum distributed algorithms [2, 10, 17, 18, 25, 27, 37]. Fraigniaud, Le Gall, Nishimura, and Paz [10], in particular, have investigated the power of distributed quantum proofs in distributed computing, which is the natural quantum version of the concept of distributed classical proofs (also called locally-checkable proofs [14] or proof-labeling schemes [23]): each node of the network receives, additionally to its input, a quantum state (called a quantum proof) from an all-powerful but untrusted party called the prover. The main result from [10] shows that there exist classical problems that can be solved by quantum protocols using quantum proofs of length exponentially smaller than in the classical case.

We present two main results about state generation and verification in the setting where an all-powerful but untrusted prover helps the nodes in a non-interactive way, and apply these results to design new quantum protocols for concrete problems studied recently in [10, 33].

## 1.1 First result and applications: State Generation with Distributed Inputs

One of the main conceptual contributions of this paper is introducing the following problem: In a network of  $r + 1$  nodes  $v_0, v_1, \dots, v_r$ , node  $v_0$  is given as input an  $n$ -qubit quantum state  $|\psi\rangle$ . The goal is to generate the quantum state  $U|\psi\rangle$  at node  $v_r$ , where  $U$  is a unitary matrix whose description is distributed over the nodes of the network. For concreteness, in this paper we focus on the case where the network is a path of length  $r$  and the nodes  $v_0, v_r$  are both extremities of the path.<sup>1</sup>

Here is the precise description of the problem. The parties  $v_0, v_1, \dots, v_r$  are the nodes of a line graph of length  $r$ : the left-end extremity is  $v_0$ , the right-end extremity is  $v_r$ , and nodes  $v_j$  and  $v_{j+1}$  are connected for  $j = 0, 1, \dots, r - 1$ . Node  $v_0$  receives as input the classical description of an  $n$ -qubit state  $|\psi\rangle$ , as a  $2^n$ -dimensional vector.<sup>2</sup> The other nodes  $v_j$  for  $j = 1, 2, \dots, r$  receive as input the description of an  $n$ -qubit unitary transformation: each node  $v_j$  receives the description of a unitary transformation  $U_j$  acting on  $n$  qubits. In this setting, the aim is to generate the quantum state

<sup>1</sup> In distributed computing it is standard to first investigate the complexity of computational problems on simple network topologies such as a path or a ring. A solution on the path can often be extended to networks of more complex topology, or be used as a building block for solving problems on network of arbitrary topology.

<sup>2</sup> Our protocol actually only requires  $v_0$  to be able to generate many copies of  $|\psi\rangle$ , and thus also works when the input is a description of a quantum circuit generating  $|\psi\rangle$ , or even a black box generating  $|\psi\rangle$ .

$$|\varphi_r\rangle := U_r \cdots U_1 |\psi\rangle$$

at the right-end extremity  $v_r$ . We call this problem  $n$ -qubit *State Generation with Distributed Inputs* on the line of length  $r$  ( $n$ -qubit  $\text{SGDI}_r$ ). Without a prover, this problem is clearly not solvable in less than  $r$  rounds of communications between neighbors (this can be seen easily by considering the case where  $U_1 = \cdots = U_r = I$ ).

We consider the setting where a prover (an all-powerful but untrusted party) helps the nodes in a non-interactive way: at the very beginning of the protocol the prover sends to node  $v_j$  a quantum state  $\rho_j$  of at most  $s_c$  qubits, for each  $j \in \{0, 1, \dots, r\}$ . Here  $s_c$  is called the certificate size of the protocol and the state  $\rho_j$  is called the certificate to  $v_j$ . The nodes then run a one-round<sup>3</sup> distributed quantum algorithm (called the verification algorithm). More precisely, the nodes first perform one round of (synchronous) communication: each node sends one quantum message of at most  $s_m$  qubits to its neighbors ( $s_m$  is called the message size of the protocol). Each node then decides to either accept or reject. Such protocols, which have been introduced and studied in [10], are called distributed Quantum Merlin-Arthur (dQMA) protocols (see Section 2 for details). Additionally, when considering dQMA protocols for  $n$ -qubit  $\text{SGDI}_r$ , we add the requirement that node  $v_r$  outputs an  $n$ -qubit quantum state at the end of the protocol.

Here is our main result:

► **Theorem 1.** *For any constant  $\varepsilon > 0$ , there exists a dQMA protocol for  $n$ -qubit  $\text{SGDI}_r$  with certificate size  $O(n^2 r^5)$  and message size  $O(nr^2)$  satisfying the following: (**completeness**) There are certificates  $\rho_0, \dots, \rho_r$  such that all the nodes accept and node  $v_r$  outputs  $|\varphi_r\rangle$  with probability 1; (**soundness**) If all the nodes accept with probability at least  $\varepsilon$ , then the output state  $\rho$  of node  $v_r$  satisfies  $\langle \varphi_r | \rho | \varphi_r \rangle \geq 1 - \varepsilon$ .*

The protocol of Theorem 1 is a dQMA protocol with perfect completeness and soundness  $\varepsilon$ . Indeed, when receiving appropriate certificates from the prover, all nodes accept with probability 1 and node  $v_r$  outputs the state  $|\varphi_r\rangle$ . On the other hand, if the state  $\rho$  is far from  $|\varphi_r\rangle$ , the soundness condition guarantees that for any certificates  $\rho_0, \dots, \rho_r$  received from the prover (including the case of entangled certificates), the probability that at least one node rejects is at least  $1 - \varepsilon$  (remember that the quantity  $\langle \varphi_r | \rho | \varphi_r \rangle$  represents the square root of the fidelity between  $|\varphi_r\rangle\langle \varphi_r|$  and  $\rho$ ).

As an application of Theorem 1, we construct a quantum protocol for a concrete computational task called Set Equality, which was introduced in Ref. [33]. Here is the formal definition over a network of arbitrary topology (represented by an arbitrary graph  $G = (V, E)$ ).

► **Definition 1** ( $\text{SetEquality}_{\ell, U}$  [33]). *Let  $\ell$  be a positive integer and  $U$  be a finite set. Each node  $u$  of a graph  $G = (V, E)$  holds two lists of  $\ell$  elements  $(a_{u,1}, \dots, a_{u,\ell})$  and  $(b_{u,1}, \dots, b_{u,\ell})$  as input, where  $a_{u,i}, b_{u,i} \in U$  for all  $i \in \{1, 2, \dots, \ell\}$ . Define  $A = \{a_{u,i} \mid u \in V, i \in \{1, 2, \dots, \ell\}\}$  and  $B = \{b_{u,i} \mid u \in V, i \in \{1, 2, \dots, \ell\}\}$ . The output of  $\text{SetEquality}_{\ell, U}$  is 1 (yes), if  $A = B$  as multisets and 0 (no) otherwise.*

Using Theorem 1 we obtain the following result:

<sup>3</sup> As in almost all prior works on (classical or quantum) distributed proofs, in this paper we consider only one-round verification algorithms.

► **Theorem 2.** *For any small enough constant  $\varepsilon > 0$ , there exists a dQMA protocol for  $\text{SetEquality}_{\ell,U}$  on the line graph of length  $r$  with completeness  $1 - \varepsilon$  and soundness  $\varepsilon$  that has certificate size  $O(r^5 \log^2(\ell r) \log^2 |U|)$  and message size  $O(r^2 \log(\ell r) \log |U|)$ .*

While Ref. [33] considered the special case of  $\text{SetEquality}_{\ell,U}$  and showed efficient distributed *interactive* protocols with small certificate and message size (see Section 1.4), no (nontrivial) classical dMA protocol (or lower bound) is known before this paper to our best knowledge. We complement the result in Theorem 2 by showing classical lower bounds and upper bounds of distributed Merlin-Arthur (dMA) protocols for  $\text{SetEquality}_{\ell,U}$ .

► **Theorem 3.** *For any dMA protocol for  $\text{SetEquality}_{\ell,U}$  on a line graph of length  $r$  with certificate size  $s_c$ , completeness  $3/4$ , and soundness  $1/4$ ,*

- *if  $|U| < \ell$ , then  $s_c = \Omega(|U| \log(\ell/|U|))$ ;*
- *if  $|U| = \Omega(\ell)$ , then  $s_c = \Omega(\ell)$ ;*
- *if  $|U| = \Omega(r\ell)$ , then  $s_c = \Omega(r\ell)$ .*

► **Theorem 4.** *There exists a dMA protocol for  $\text{SetEquality}_{\ell,U}$  on a line graph of length  $r$  with completeness 1 and soundness 0 whose certificate size and message size are both  $O(\min\{r\ell \log |U|, |U| \log(r\ell)\})$ .*

Although the dependence in  $r$  is worse than in the classical dMA protocol of Theorem 4, the dependence of the dQMA protocol of Theorem 2 in  $\ell$  (the number of elements each node receives) and  $|U|$  (the size of the universal set) are polylogarithmic. On the other hand, in classical case, we have linear lower bounds with respect to  $\ell$  and  $|U|$  as in Theorem 3. Therefore Theorem 2 gives a significant improvement for sufficiently large  $\ell$  and  $|U|$ . This assumption about the input parameters seems reasonable when considering applications similar to those of the dQMA protocol for the equality problem proposed in Ref. [10]. Note that our bounds of classical certificate size in Theorem 3 and Theorem 4 are tight up to  $\text{poly} \log(\ell, |U|, r)$  factors when  $|U| < \ell$  or  $|U| = \Omega(r\ell)$ .

## 1.2 Second result and applications: EPR-pairs generation and LOCC dQMA protocols

Our second contribution is a protocol, based on a recent work by Zhu and Hayashi [41], to create EPR-pairs between adjacent nodes of a network without quantum communication in the same setting as above, where a prover helps the nodes in a non-interactive way. As an application of this protocol, we prove a general result showing how to convert any dQMA protocol on an arbitrary network into another dQMA protocol where the verification algorithm uses only classical communication (instead as quantum communication, as allowed in the definition of dQMA protocols and used in all dQMA protocols of Ref. [10] and Theorems 1 and 2 above).

More precisely, we say a dQMA protocol is an LOCC (Local Operation and Classical Communication) dQMA protocol if the verification algorithm can be implemented only by local operations at each node and classical communication between neighboring nodes (i.e., no quantum communication is allowed). Our protocol for generating EPR-pairs enables us to show the following theorem:

► **Theorem 5.** *For any constants  $p_c$  and  $p_s$  such that  $0 \leq p_s < p_c \leq 1$ , let  $\mathcal{P}$  be a dQMA protocol for some problem on a network  $G$  with completeness  $p_c$ , soundness  $p_s$ , certificate size  $s_c^{\mathcal{P}}$  and message size  $s_m^{\mathcal{P}}$ . For any small enough constant  $\gamma > 0$ , there exists an LOCC dQMA protocol  $\mathcal{P}'$  for the same problem on  $G$  with completeness  $p_c$ , soundness  $p_s + \gamma$ , certificate size  $s_c^{\mathcal{P}'} + O(d_{\max} s_m^{\mathcal{P}} s_{tm}^{\mathcal{P}'})$ , and message size  $O(s_m^{\mathcal{P}} s_{tm}^{\mathcal{P}'})$ , where  $d_{\max}$  is the maximum degree of  $G$ , and  $s_{tm}^{\mathcal{P}'}$  is the total number of qubits sent in the verification stage of  $\mathcal{P}'$ .*

As an application of Theorem 5, we consider the equality problem studied in Ref. [10]. In this problem, denoted  $\text{EQ}_n^t$ , a collection of  $n$ -bit strings  $x_1, x_2, \dots, x_t$  is given as input to  $t$  specific nodes  $u_1, u_2, \dots, u_t$  (called terminals) of an arbitrary network  $G = (V, E)$  as follows: node  $u_i$  receives  $x_i$ , for  $i \in \{1, 2, \dots, t\}$ . The goal is to check whether the  $t$  strings are equal, i.e., whether  $x_1 = \dots = x_t$ . By applying Theorem 5 to the main result in Ref. [10] (a dQMA protocol for  $\text{EQ}_n^t$  with certificate size  $O(tr^2 \log n)$  and message size  $O(tr^2 \log(n+r))$ ), we obtain the following corollary:

► **Corollary 1.** *For any small enough constant  $\varepsilon > 0$ , there is an LOCC dQMA protocol for  $\text{EQ}_n^t$  with completeness 1, soundness  $\varepsilon$ , certificate size  $O(d_{\max}|V|t^2r^4 \log^2(n+r))$  and messages size  $O(|V|t^2r^4 \log^2(n+r))$ , where  $r$  is the radius of the set of the  $t$  terminals and  $|V|$  is the number of nodes of the network  $G = (V, E)$ .*

We can also apply Theorem 5 to the dQMA protocol of Theorem 2, leading to the following corollary:

► **Corollary 2.** *For any small enough constant  $\varepsilon > 0$ , there is an LOCC dQMA protocol for  $\text{SetEquality}_{\ell, U}$  on the line graph of length  $r$  with completeness  $1 - \varepsilon$ , soundness  $\varepsilon$ , certificate size  $O(r^5 \log^2(\ell r) \log^2 |U|)$  and messages size  $O(r^5 \log^2(\ell r) \log^2 |U|)$ .*

Note that these LOCC dQMA protocols still have good dependence in the main parameters we are interested in: the parameter  $n$  for  $\text{EQ}_n^t$  (for which the dependence is still exponentially better than any classical dMA protocols) and the parameters  $\ell$  and  $|U|$  for  $\text{SetEquality}_{\ell, U}$  (for which the dependence is still exponentially better than any classical dMA protocols, due to Theorem 3).

### 1.3 Overview of our proofs

To explain the proof idea of Theorem 1, we only consider the simplified case  $U_1 = \dots = U_r = I$ . The general case can be proved similarly by a slightly more complicated analysis.

The dQMA protocol to prove Theorem 1 is based on the dQMA protocol on the line of length  $r$  by Fraigniaud et al. [10]. In the setting of Ref. [10], the left-end extremity  $v_0$  has an  $n$ -bit string  $x$ , the right-end extremity  $v_r$  has an  $n$ -bit string  $y$ , and the other intermediate nodes have no input. The goal is to verify whether  $x = y$ . The dQMA protocol in Ref. [10] checks whether the fingerprint state  $|\psi_0\rangle = |\psi_x\rangle$  [5] prepared by  $v_0$  is equal to the fingerprint state  $|\psi_r\rangle = |\psi_y\rangle$  prepared by  $v_r$  ( $x = y$ ), or  $|\psi_0\rangle$  is almost orthogonal to  $|\psi_r\rangle$  ( $x \neq y$ ). For this, node  $v_j$  ( $2 \leq j \leq r-1$ ) receives a subsystem whose reduced state is  $\rho_j$  as a certificate from the prover. At the verification stage, any node (except for  $v_r$ ) chooses keeping its certificate by itself, or sending it to the right neighboring node with probability  $1/2$  to check if the reduced states of the two neighboring nodes,  $\rho_j$  and  $\rho_{j+1}$ , are close, which can be checked by the SWAP test [5]. If  $x = y$ , then the prover can send  $|\psi_0\rangle$  ( $= |\psi_r\rangle$ ) for every intermediate node to pass all the SWAP tests done at the verification stage, which means accept. Otherwise, the SWAP test done at some node rejects with a reasonable probability since  $|\psi_x\rangle$  is very far from  $|\psi_y\rangle$ , and hence the distance between  $\rho_j$  and  $\rho_{j+1}$  should be far at some  $j$ .

Now the case that  $U_1 = \dots = U_r = I$  (which means that all nodes except  $v_0$  have no input) in the setting of  $\text{SGDI}_r$  (then the goal state  $|\varphi_r\rangle$  at  $v_r$  is the same as the state  $|\psi\rangle$  of  $v_0$ ) is similar to the setting of Ref. [10], except that  $v_r$  also has no input. The difficulty is that  $v_r$  has no state that can be generated by itself, and thus the analysis of Ref. [10] cannot be used as it is.

To overcome this difficulty, we utilize an idea from the verification of graph states [15, 32], in particular, the idea by Morimae, Takeuchi, and Hayashi [32]. They used the following basic idea for their protocol in order to verify an arbitrary graph state  $|G\rangle$  sent from the prover (or prepared by a malicious party): (i) the verifier receives  $(m + k + 1)$  subsystems, in which each subsystem ideally contains  $|G\rangle$ , from the prover; (ii) the verifier chooses  $m$  subsystems uniformly at random, and discards them; (iii) the verifier chooses one subsystem, and some test that  $|G\rangle$  should pass (stabilizer test) is done for each of the remaining  $k$  subsystems; and (iv) if all the tests passed, the chosen subsystem in (iii) should be close to  $|G\rangle$ , which is proved by using a quantum de Finetti theorem with some measurement condition [28] (exponentially better in the dimension of the subsystem than the standard quantum de Finetti theorem [6]). Note that (ii) and (iii) are necessary since the assumption that the total system is permutation-invariant is needed to apply the quantum de Finetti theorem.

Our protocol applies the idea of Ref. [32] to the verification protocol of Ref. [10] explained above. Namely, the parties  $v_1, v_2, \dots, v_r$  first receives  $(m + k + 1)$  subsystems, where each subsystem ideally contains  $|\psi\rangle^{\otimes r}$ , sent from the prover. For  $k$  subsystems that are randomly chosen, we apply the verification protocol of Ref. [10]. Actually, we have a subtle problem with the corresponding steps of (ii) and (iii) in the idea of Ref. [32], since  $v_0, v_1, \dots, v_r$  do not have any shared randomness, and thus those steps cannot be implemented jointly. Fortunately, this problem can be overcome since the permutation-invariant property is satisfied by the random permutations of  $(m + k + 1)$  subsystems on *each* party.

The dQMA protocol for Theorem 2 is based on the distributed interactive protocol by Naor, Parter, and Yegorov [33] using shared randomness<sup>4</sup>. In our setting (line of length  $r$ ), the distributed interactive protocol of Ref. [33] is as follows with two polynomials  $\alpha_j(x) := \prod_i (x - a_{j,i})$  and  $\beta_j(x) := \prod_i (x - b_{j,i})$ : with shared randomness  $s$  (taken from a large field), (i)  $v_0$  prepares  $A_0(s) := \alpha_0(s)$  and  $B_0(s) := \beta_0(s)$ ; (ii)  $v_j$  ( $j = 1, 2, \dots, r$ ) ideally receives  $A_j(s) := \alpha_0(s) \cdots \alpha_j(s)$  and  $B_j(s) := \beta_0(s) \cdots \beta_j(s)$  from the prover; (iii)  $A_j(s) = \alpha_j(s)A_{j-1}(s)$  and  $B_j(s) = \beta_j(s)B_{j-1}(s)$  are checked for consistency by communication from  $v_{j-1}$  to  $v_j$ . We can see that when  $A = B$ ,  $A_r(s) = B_r(s)$  for any  $s$ , and thus this protocol accepts with probability 1 by the ideal certificates from the prover, while when  $A \neq B$ ,  $A_r(s) \neq B_r(s)$  for most of  $s$ , and thus some node rejects with reasonable probability.

Actually, neither interaction nor shared randomness is available in our setting. Instead, we reduce the protocol by Naor et al. to  $\text{SGDI}_r$  with  $|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle$  where  $|\psi_A\rangle = \sum_s |s\rangle |\alpha_0(s)\rangle$ , and  $|\psi_B\rangle = \sum_s |s\rangle |\beta_0(s)\rangle$ , and  $U = U_{j,A} \otimes U_{j,B}$ , where  $U_{j,A}$  roughly<sup>5</sup> maps  $|s\rangle |t\rangle$  to  $|s\rangle |\alpha_j(s)t\rangle$  ( $j = 1, 2, \dots, r$ ) and  $U_{j,B}$  roughly maps  $|s\rangle |t\rangle$  to  $|s\rangle |\beta_j(s)t\rangle$  ( $j = 1, 2, \dots, r$ ). Then, Theorem 1 guarantees that  $v_r$  receives  $\sum_s |s\rangle |A_r(s)\rangle$  and  $\sum_s |s\rangle |B_r(s)\rangle$  with high fidelity as long as every node accepts with at least the probability guaranteed by Theorem 1. The SWAP test between these at  $v_r$  checks if  $A = B$  with high probability.

For the classical lower bound of  $\text{SetEquality}_{\ell,U}$  in Theorem 3, we utilize the lower bound for  $\text{EQ}_n^2$  of [10]. Ref. [10] showed that for any classical protocol for  $\text{EQ}_n^2$  on the line graph, at least one internal node requires a certificate of linear size. We show that  $\text{EQ}_n^2$  can be reduced to  $\text{SetEquality}_{\ell,U}$  in three cases depending on the size of  $U$ . Here we explain the simplest case:  $|U| = \Omega(\ell)$ . For a line graph with the left-end extremity  $v$  and the right-end extremity  $v'$ , let  $x = x_1 x_2 \cdots x_n$  be the input of  $\text{EQ}_n^2$  for  $v$  and  $y = y_1 y_2 \cdots y_n$  be the input of  $\text{EQ}_n^2$  for  $v'$ . Then we consider an injection  $f$  from  $\{0, 1\}^n$  to the set of  $3\ell$ -bit strings with

<sup>4</sup> While there is no shared randomness in their setting, shared randomness can be simulated by two interactions between the prover and the verifier.

<sup>5</sup> We actually need some modifications for  $U_{j,A}$  to be unitary.

Hamming weight  $\ell$  such that the input list  $(a_{v,1}, \dots, a_{v,\ell})$  of  $\text{SetEquality}_{\ell,U}$  for  $v$  includes the  $j$ -th element of the universal set  $U$  for  $|U| > 3\ell$  if and only if  $f(x)_j = 1$ , and the input list  $(b_{v',1}, \dots, b_{v',\ell})$  of  $\text{SetEquality}_{\ell,U}$  for  $v'$  includes the  $j$ -th element of the universal set  $U$  if and only if  $f(y)_j = 1$ . Now these two sets are identical if and only if  $x = y$ , which means a reduction from  $\text{EQ}_n^2$  to  $\text{SetEquality}_{\ell,U}$  for  $\ell = \Theta(n)$ . We thus get a lower bound of  $\Omega(\ell)$  from the  $\Omega(n)$  lower bound of  $\text{EQ}_n^2$  mentioned above.

The classical upper bound of  $\text{SetEquality}_{\ell,U}$  in Theorem 4 is fairly simple: the prover can send all of inputs  $A$  and  $B$  to each node to achieve the first upper bound  $O(r\ell \log |U|)$ . For the second upper bound  $O(|U| \log(r\ell))$ , the node  $v_i$  on the line graph  $\{v_0, \dots, v_r\}$  is given the information of inputs of  $v_j, j \in \{0, \dots, i-1\}$  as the certificate in the form of the number of each element of  $U$  in the corresponding inputs.

The basic proof idea of Theorem 5 is standard: we replace one qubit communicated between any two nodes  $u$  and  $v$  by two bits using quantum teleportation [4], assuming that they share an EPR pair  $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  sent from the prover. The problem is that the prover may be malicious, and  $u$  and  $v$  should then verify that the pair sent from the prover is  $|\Phi^+\rangle$ . In order to obtain  $|\Phi^+\rangle$  with high fidelity, we actually ask the prover to send  $N+1$  copies of the EPR pairs. An honest prover will send the state  $|\Phi^+\rangle^{\otimes(N+1)}$ , but a malicious prover may naturally send an arbitrary state. Nodes  $u$  and  $v$  use  $N$  among the  $N+1$  pairs for the verification. If the verification succeeds, they are guaranteed that the remaining pair has high fidelity with  $|\Phi^+\rangle$ .

This type of verification of  $|\Phi^+\rangle$  in an adversarial scenario by the malicious prover was considered in a remarkable work by Zhu and Hayashi [41]. Extending the previous result [34] in a less adversarial scenario, they showed that by taking  $N = O(\frac{1}{\varepsilon} \log(\frac{1}{\delta}))$ , if the verification test succeeds with probability at least  $\delta$ , the state  $\sigma$  of the last pair has a high fidelity with  $|\Phi^+\rangle$  such that  $\langle \Phi^+ | \sigma | \Phi^+ \rangle \geq 1 - \varepsilon$ . Furthermore, the measurements in their verification protocol (essentially the same as those in Ref. [34]) are local, namely, they do not need any entangled measurement between the two qubits of each pair.

Now the proof idea of Theorem 5 uses the verification protocol of Ref. [41] in our setting. To do so, we first observe that the amount of classical communication needed between  $u$  and  $v$  can be upper-bounded by  $O(N)$  (which is the same as the certificate size from the prover), by rewriting the protocol of Ref. [41] with a slight modification in our setting. Then we replace the quantum bits sent among the nodes in the original dQMA protocol  $\mathcal{P}$  by classical communication. However, it needs not only a single EPR pair but a lot of EPR pairs to be verified. Thus, we need further analysis to convert  $\mathcal{P}$  into an LOCC dQMA protocol and to evaluate the message size of classical communication and the certificate size.

## 1.4 Related work

The concept of *distributed Merlin-Arthur protocols* (dMA), which is very similar to the concept of *randomized proof-labeling schemes* [12] was introduced by [11] as a randomized version of locally checkable proofs (LCPs). In a dMA protocol, as in LCPs, the prover assigns each node a short certificate. The nodes then perform a 1-round distributed algorithm, i.e., exchange messages with their neighbors through incident edges. The difference is that in dMA, this algorithm can be a randomized algorithm, instead of a deterministic algorithm as in LCPs. This randomization is helpful to reduce the size of certificates for some problems.

The recent paper [22] introduced the interactive extension of dMA, *distributed interactive proofs*, in which the prover and the verifier can perform more interaction. They showed that interaction is also useful to reduce the size of certificates. This concept has recently been explored in depth by several studies: distributed interactive proofs that utilize quantum certificates [26], the role of shared and private randomness [7, 30], and more efficient protocols

for concrete problems [19, 31, 33]. In particular, [33] introduced  $\text{SetEquality}_{\ell,U}$ , which is one of the problems we study in this paper, and showed efficient interactive protocols for  $\text{SetEquality}_{\ell,U}$  when  $\ell = |V|$  and  $|U| = O(|V|)$  that require two interactions between the prover and the verifier with certificate size<sup>6</sup>  $O(\log |V|)$ , and five interactions between the prover and the verifier with certificate size  $O(\log \log |V|)$ .

The technique we used in this paper from Refs. [32, 41] belongs to a broad and hot topic called “state certification (state verification)” [21, 38]. One conceptual contribution of this paper is providing the first concrete example of the effective use of these techniques for quantum distributed verification.

## 2 dQMA protocols

We consider a decision problem on a connected graph (called the network)  $G = (V, E)$ , where  $t$  inputs  $x_1, x_2, \dots, x_t$  are assigned to  $t$  nodes  $v_1, v_2, \dots, v_t \in V$ . We interpret the decision problem as a Boolean function  $f$ , where  $f(x_1, x_2, \dots, x_t) = 1$  is interpreted as “yes” and  $f(x_1, x_2, \dots, x_t) = 0$  is interpreted as “no”.

The concept of distributed quantum Merlin-Arthur (dQMA) protocols on a graph  $G = (V, E)$  is a quantum version of the concept of distributed Merlin-Arthur (dMA) protocols. The aim of a dMA protocol is to verify whether  $f(x_1, x_2, \dots, x_t) = 1$  or not. As briefly explained in Section 1.1, the nodes of  $G$  (which correspond to the verifier) first receive a message from a powerful but possibly malicious party (the prover). The nodes then enter a verification phase, in which they communicate together (but do not communicate with the prover anymore). The communication is possible only if two nodes are connected: each node can send one message to each of its neighbors. In the case of dQMA protocols, the only difference is that the message from the prover and the communication among the nodes may be quantum. Note that neither randomness nor entanglement are shared among the nodes in advance.

Formally, in a dQMA protocol  $\mathcal{P}$  on  $G = (V, E)$ , each node  $u \in V$  first receives a quantum register  $M_u$  from the prover. Then the nodes move to the verification stage, which consists of the following steps: (i)  $u$  applies a local quantum (or classical) operation on the composite system of  $M_u$  and its private register  $V_u$ ; (ii)  $u$  sends a quantum (or classical) register  $M_{uv}$  to any neighboring node  $v$ , and (iii)  $u$  applies a local quantum (or classical) operation on  $M_u$ ,  $V_u$ , and  $\otimes_{v \in N(u)} M_{vu}$ , and either accepts or rejects (we call this the decision of  $u$ ), where  $N(u)$  denotes the set of nodes that are neighbors of  $u$ . When local operations at each node and communication among the nodes in the verification stage are classical, the dQMA protocol is called *LOCC (Local Operation and Classical Communication)*.

The two main complexity measures of  $\mathcal{P}$  are the certificate size and the message size. The certificate size of  $\mathcal{P}$ , denoted as  $s_c^{\mathcal{P}}$ , is the maximum number of qubits that are sent to each node from the prover, that is,  $s_c^{\mathcal{P}} := \max_{u \in V} |M_u|$ , where  $|R|$  denotes the number of qubits of  $R$ . The message size of  $\mathcal{P}$ , denoted as  $s_m^{\mathcal{P}}$ , is the maximum number of qubits sent on edges of  $G$ , namely,  $s_m^{\mathcal{P}} := \max_{(u,v) \in E} (|M_{uv}| + |M_{vu}|)$ .

A dQMA protocol  $\mathcal{P}$  for a decision problem  $f$  on  $G$  with completeness  $p_c$  and soundness  $p_s$  is defined as a dQMA protocol satisfying the following two conditions:

**(completeness)** If  $f(x_1, x_2, \dots, x_t) = 1$ , there exists some quantum state  $|\chi\rangle$  on  $M := \otimes_{u \in V} M_u$  such that  $\Pr[\text{all nodes accept}] \geq p_c$ ;

**(soundness)** If  $f(x_1, x_2, \dots, x_t) = 0$ , for any quantum state  $|\chi\rangle$  on  $M$ ,  $\Pr[\text{all nodes accept}] \leq p_s$ .

<sup>6</sup> For  $\text{SetEquality}_{\ell,U}$ , the certificate size of their protocol can be written as  $O(\log |U| + \log(\ell|V|))$ .

In this paper, we consider the problem of generating a quantum state  $|\varphi\rangle$  on a network  $G = (V, E)$ . In this problem, some initially specified nodes  $w_1, \dots, w_\kappa$  not only make their decisions (accept or reject) but also output the quantum state  $|\varphi\rangle$  jointly (if they accept). In our specific problem, the  $n$ -qubit  $\text{SGDI}_r$ , all nodes of the line graph with nodes  $v_0, v_1, \dots, v_r$  have an input ( $v_0$  has a classical description of  $|\psi\rangle$  and  $v_j$  for  $j = 1, 2, \dots, r$  has a classical description of  $U_j$ ),  $|\varphi\rangle = |\varphi_r\rangle$  ( $:= U_r \cdots U_1 |\psi\rangle$ ),  $\kappa = 1$ , and  $w_1 = v_r$ .

In a dQMA protocol for the problem of generating  $|\varphi\rangle$  on  $G$ , the completeness and soundness conditions are slightly different from the case of decision problems. For our purpose we actually only need to discuss perfect-completeness protocols. We say that the dQMA protocol has perfect completeness and  $(\delta, \varepsilon)$ -soundness if the following completeness and soundness are satisfied:

**(completeness)** There exists a quantum state  $|\chi\rangle$  on  $M$  such that

$$\Pr[\text{all nodes accept and } w_1, \dots, w_\kappa \text{ output } |\varphi\rangle \text{ jointly}] = 1;$$

**(soundness)** If all nodes accept with probability at least  $\delta$ , then the output  $\tilde{\rho}$  of  $w_1, \dots, w_\kappa$  (under the condition that all nodes accept) satisfies

$$\langle \varphi | \tilde{\rho} | \varphi \rangle \geq 1 - \varepsilon.$$

The soundness condition is regarded as a kind of hypothesis testing (i.e., if the verifier's test passes with probability greater than a threshold, then the state would be close to the ideal one). A similar completeness-soundness condition is used for the interactive proofs for synthesizing quantum states [35].

### 3 dQMA Protocol for State Generation with Distributed Inputs

In this section we present our dQMA protocol for the  $n$ -qubit State Generation with Distributed Inputs over the line of length  $r$  ( $n$ -qubit  $\text{SGDI}_r$ ) and prove Theorem 1.

#### 3.1 dQMA protocol for SGDI

The following is our dQMA protocol for  $n$ -qubit  $\text{SGDI}_r$ .

**Protocol  $\mathcal{P}_{\text{SGDI}}$ :** Let  $k = 144cr^{2+\eta}$  and  $m = 2cnk^2(r+1)^{1+\eta}$  for any constant  $c > 0$  and any small constant  $\eta \geq 0$ .

1.  $v_0$  prepares  $(m+k+1)$  copies of  $|\psi\rangle$  in  $n$ -qubit registers  $R_{0,j}$  ( $j = 1, 2, \dots, m+k+1$ ).
2. The prover sends each  $v_l$ , where  $l = 1, 2, \dots, r$ ,  $(m+k+1)$   $n$ -qubit registers  $R_{l,1}, R_{l,2}, \dots, R_{l,m+k+1}$ .
3. Each  $v_l$  ( $l = 1, 2, \dots, r$ ) permutes the  $(m+k+1)$  registers  $R_{l,1}, R_{l,2}, \dots, R_{l,m+k+1}$  by a permutation  $\pi$  on  $\{1, 2, \dots, m+k+1\}$  taken uniformly at random, and renames  $R_{l,j} := R_{l,\pi(j)}$ .
4. The parties  $v_0, v_1, \dots, v_r$  implement the following subprotocol  $\mathcal{P}_{\text{SGDIV}}$  (a modification of the verification steps in Ref. [10]) on registers  $R_{0,j}, R_{1,j}, \dots, R_{r,j}$  for each  $j = 2, 3, \dots, k+1$  in order. If some party rejects for some  $j$ , the protocol rejects.
5.  $v_r$  outputs  $R_{r,1}$ .

**Protocol  $\mathcal{P}_{\text{SGDIV}}$ :** Assume that  $v_0$  has  $|\psi\rangle$  on  $n$ -qubit register  $R_0$ , and  $v_l$  ( $l = 1, 2, \dots, r$ ) receives  $n$ -qubit register  $R_l$ .

1. For every  $j = 0, 1, \dots, r - 1$ , party  $v_j$  chooses a bit  $b_j$  uniformly at random, and sends its register  $R_j$  to the right neighbor  $v_{j+1}$  whenever  $b_j = 0$ .
2. For every  $j = 1, 2, \dots, r$ , if  $v_j$  receives a register from the left neighbor  $v_{j-1}$ , and if  $b_j = 1$ , then  $v_j$  applies  $U_j$  on register  $R_{j-1}$ , and performs the SWAP test on the registers  $(R_{j-1}, R_j)$ , and accepts or rejects accordingly; Otherwise,  $v_j$  accepts.

We can show the following theorem, which induces Theorem 1 by a special case with  $\eta = 0$ .

► **Theorem 6.** *Protocol  $\mathcal{P}_{\text{SGDI}}$  has perfect completeness and  $(\frac{1}{(cr^\eta)^{1/4}}, \frac{1}{(cr^\eta)^{1/4}})$ -soundness. The certificate size of  $\mathcal{P}_{\text{SGDI}}$  is  $O(n^2r^{5+3\eta})$  and the message size is  $O(nr^{2+\eta})$ .*

### 3.2 Proof of Theorem 6

We can see that the certificate size of  $\mathcal{P}_{\text{SGDI}}$  is  $(m + k + 1)n = O(n^2r^{5+3\eta})$  from step 2 of  $\mathcal{P}_{\text{SGDI}}$ . Since  $\mathcal{P}_{\text{SGDI}}$  implements  $\mathcal{P}_{\text{SGDIV}}$   $(k + 1)$  times, and the message size of  $\mathcal{P}_{\text{SGDI}}$  is  $O(n)$ , the message size of  $\mathcal{P}_{\text{SGDI}}$  is  $O(nk) = O(nr^{2+\eta})$ .

The completeness clearly holds: since the prover honestly sends

$$|\varphi_l\rangle := U_l \cdots U_1 |\psi\rangle$$

as the content of  $R_{l,j}$  for each  $j \in \{1, 2, \dots, m + k + 1\}$  and then all the SWAP tests in  $\mathcal{P}_{\text{SGDIV}}$  accept with probability 1.

The proof of the soundness can be found in the full version.

## 4 Application: dQMA Protocol for Set Equality

In this section we prove Theorem 2 by constructing a protocol for  $\text{SetEquality}_{\ell,U}$  based on the protocol for  $\text{SGDI}_r$  developed in Section 3.

**Proof of Theorem 2.** We consider  $\text{SetEquality}_{\ell,U}$  (Definition 1) for the line graph of length  $r$  with nodes  $v_0, v_1, \dots, v_r$ , where  $v_j$  has  $a_{j,1}, \dots, a_{j,\ell}$  and  $b_{j,1}, \dots, b_{j,\ell}$ . Let  $\alpha_j(s) := \prod_{i \in \{1, 2, \dots, \ell\}} (s - a_{j,i})$  and  $\beta_j(s) := \prod_{i \in \{1, 2, \dots, \ell\}} (s - b_{j,i})$  for each  $j \in \{0, 1, \dots, r\}$ . We identify  $a_{u,i}, b_{u,i}$  as elements in a finite field  $\mathbb{F}$  with size  $|\mathbb{F}| \geq \tilde{c}\ell(r + 1)2^{\log|U|}$  for some (sufficiently large) constant  $\tilde{c} > 0$ . Our goal is the same as of the interactive protocol of [33] – the node  $v_r$  checks if two polynomials  $p_A(s) := \prod_{j \in \{0, 1, \dots, r\}} \alpha_j(s)$  and  $p_B(s) := \prod_{j \in \{0, 1, \dots, r\}} \beta_j(s)$  take the same value for uniform randomly chosen  $s \in \mathbb{F}$ , where  $s$  is distributed by the interaction. In order to implement this idea in a non-interactive way, we utilize the framework of  $\text{SGDI}$  as follows: Let

$$|\psi\rangle := \frac{1}{\sqrt{|\mathbb{F}|}} \sum_{s \in \mathbb{F}} |s\rangle |\alpha_0(s)\rangle$$

be the initial state that the node  $v_0$  can locally produce. Each node  $v_j, j \in \{1, 2, \dots, r\}$  has a unitary transformation  $U_j$  which maps  $|s\rangle |\alpha_{j-1}(s)\rangle$  to  $|s\rangle |\alpha_j(s)\rangle$  (to be precise, we also need to deal with the case where  $\alpha_j(s)$  is a zero polynomial). Using the protocol for  $\text{SGDI}$  in Theorem 1, the node  $v_r$  outputs the state  $|\varphi_r\rangle = U_r \cdots U_1 |\psi\rangle$ , which is the uniform

superposition of  $|s\rangle|p_A(s)\rangle$  over all  $s \in \mathbb{F}$ . Similarly,  $v_r$  also outputs the uniform superposition of  $|s\rangle|p_B(s)\rangle$  over all  $s \in \mathbb{F}$ . Finally,  $v_r$  does the SWAP test for these states, which accepts with high probability if and only if two polynomials are identical (i.e.,  $A = B$  as multisets) since the two polynomials  $p_A(s)$  and  $p_B(s)$  take different values for most  $s \in \mathbb{F}$  if  $A \neq B$ . In the full version we complete the proof by describing the details of the protocol (which we denote  $\mathcal{P}_{\text{seteq}}$ ) and analyzing it rigorously. ◀

#### 4.1 Classical bounds for SetEquality $_{\ell,U}$

Here we prove the classical lower bounds shown in Theorem 3. The proof of Theorem 4 is deferred to the full version.

**Proof of Theorem 3.** In order to prove our lower bounds for SetEquality $_{\ell,U}$ , we utilize reductions from EQ $_n^2$  to SetEquality $_{\ell,U}$ , then apply the following lower bound of EQ $_n^2$  that appears in [10].

► **Lemma 1** (Theorem 9 of [10]). *Let  $r \geq 3$  be a positive integer. Consider an instance of EQ $_n^2$  where the two nodes  $v_0$  and  $v_r$  on a line graph  $v_0, \dots, v_r$  are provided with inputs  $x \in \{0, 1\}^n$  and  $y \in \{0, 1\}^n$ . Then, for any dMA protocol that solves EQ $_n^2$  for this instance with completeness  $1-p$  and soundness  $1-2p-\varepsilon$  for any  $p, \varepsilon > 0$ , there exists  $i \in \{1, \dots, r-1\}$  such that the certificate size of  $v_i$  is  $\Omega(n)$ .*

We show three different reductions depending on the size of  $|U|$ . Due to space constraints, we only show the simplest case:  $|U| = \Omega(\ell)$ . The reductions for the other two cases are deferred to the full version.

Let  $\mathcal{P}$  be a dMA protocol for SetEquality $_{\ell,U}$  with the certificate size  $s_c$  which appears in the statement of the theorem. We consider the following instance of EQ $_n^2$  on a line graph  $v_0, \dots, v_r$  of length  $r \geq 3$ : the node  $v_0$  is provided with  $x$ , and the node  $v_r$  is provided with  $y$ . Let  $\ell$  be the minimum integer satisfying  $\binom{3\ell}{\ell} \geq 2^n$ . Since  $\binom{3\ell}{\ell} = 2^{3\ell H(1/3) - O(\log \ell)}$  where  $H(\cdot)$  is the binary entropy function, we have  $\ell = \Theta(n)$ . Let  $S = \{s \in \{0, 1\}^{3\ell} : |s| = \ell\}$  be the set of  $3\ell$ -bit strings so that  $|S| = \binom{3\ell}{\ell}$ . We arbitrarily choose one injection  $f : \{0, 1\}^n \rightarrow S$  (this kind of injection exists since we have  $|S| \geq 2^n$ ). The network constructs the following instance of SetEquality $_{\ell,U}$  for the universal set  $U = \{0, 1, 2, \dots, 3\ell\}$  without communication:

- The inputs  $x$  and  $y$  are converted to  $f(x), f(y) \in S$ . Let  $X = \{i : f(x)_i = 1\}$  and  $Y = \{i : f(y)_i = 1\}$  be two sets of  $\ell$  elements from the universal set  $\{1, 2, \dots, 3\ell\}$ .  $X$  and  $Y$  are regarded as the inputs  $(a_{v_0,1}, \dots, a_{v_0,\ell})$  and  $(b_{v_r,1}, \dots, b_{v_r,\ell})$  of SetEquality $_{\ell,U}$ . Furthermore, we set  $(b_{v_0,1}, \dots, b_{v_0,\ell}) = (a_{v_r,1}, \dots, a_{v_r,\ell}) = (0, \dots, 0)$ .
- The inputs to each internal node  $v_1, \dots, v_{r-1}$  are set to  $(0, \dots, 0), (0, \dots, 0)$ .

Now the set  $A$  and  $B$  of this instance of SetEquality $_{\ell,U}$  are identical as multisets if and only if  $f(x) = f(y)$ . Since  $f$  is an injection, we have  $f(x) = f(y) \Leftrightarrow x = y$  and thus the output of SetEquality $_{\ell,U}$  on this instance is identical to that of EQ $_n^2$  on the input  $x$  and  $y$ . Now we can use the protocol  $\mathcal{P}$  to solve EQ $_n^2$  for  $n = \Theta(\ell)$ . Thus by Lemma 1, we have  $s_c = \Omega(\ell)$ . ◀

## 5 Conversion of dQMA protocols into LOCC dQMA protocols

In this section we show how to create an EPR pair  $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  between two parties without quantum communication in the setting where a prover helps the nodes in a non-interactive way. Our protocol is based on the verification protocol of the EPR pair in the adversarial setting proposed by Zhu and Hayashi [41] (see also [39, 40]), who showed that a verifier  $V$  can check whether a two-qubit state sent from a (possibly malicious) prover is  $|\Phi^+\rangle$ .

The following is the verification protocol given in Ref. [41].

**Protocol  $\mathcal{P}_{\text{ZH}}$ :** Let  $R_1, R_2, \dots, R_N, R_{N+1}$  be  $(N+1)$  two-qubit registers from the prover. Here,  $|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ ,  $|-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ ,  $|+\prime\rangle := \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$  and  $|-\prime\rangle := \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$ .

1. Perform a random permutation  $\pi$  on the  $(N+1)$  two-qubit registers, and rename  $R_j := R_{\pi(j)}$  for  $j = 1, 2, \dots, N+1$ .
2. For each  $j = 1, 2, \dots, N$ , the verifier  $V$  does one of the following three POVMs on register  $R_j$  with probability  $1/3$  for each:
  - $M_1 = \{E_1, I - E_1\}$  with  $E_1 = |00\rangle\langle 00| + |11\rangle\langle 11|$ .
  - $M_2 = \{E_2, I - E_2\}$  with  $E_2 = |++\rangle\langle ++| + |--\rangle\langle --|$ .
  - $M_3 = \{E_3, I - E_3\}$  with  $E_3 = |+\prime-\prime\rangle\langle +\prime-\prime| + |-\prime+\prime\rangle\langle -\prime+\prime|$ .
3. Reject if the second components in the POVMs are obtained. Otherwise, the test passes and outputs  $R_{N+1}$ .

Ref. [41] describes  $E_1 = \frac{I+Z^{\otimes 2}}{2}$ ,  $E_2 = \frac{I+X^{\otimes 2}}{2}$  and  $E_3 = \frac{I-Y^{\otimes 2}}{2}$ , where

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \text{ and } Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

while we rewrite them as above (which is a similar expression to the protocol in Ref. [34]) since it would be easy to see for our purpose. Importantly, step 2 implements the POVM  $\{\Omega, I - \Omega\}$  on  $R_j$  with  $\Omega = \frac{2}{3}|\Phi^+\rangle\langle\Phi^+| + \frac{1}{3}I$  for each  $j$ , but it is implemented by local measurements on each qubit of  $R_j$ .

The following result was shown for the protocol  $\mathcal{P}_{\text{ZH}}$  in Ref. [41].

► **Theorem 7 (Zhu-Hayashi).** *There is a number  $N = O(\frac{1}{\varepsilon} \log(\frac{1}{\delta}))$  such that if the test passed with probability at least  $\delta$ , then the output state  $\tilde{\sigma}$  of  $\mathcal{P}_{\text{ZH}}$  (under the condition that the test passes) satisfies  $\langle\Phi^+|\tilde{\sigma}|\Phi^+\rangle \geq 1 - \varepsilon$ .*

The protocol  $\mathcal{P}_{\text{ZH}}$  uses only local measurements, and thus, it can be used for verifying the sharing of an EPR pair by two parties who only use local operations and classical communication (LOCC).

Let  $V_1$  and  $V_2$  be neighboring parties who expect to receive  $|\Phi^+\rangle$  jointly from the prover. The following protocol is a simple implementation of  $\mathcal{P}_{\text{ZH}}$  with LOCC by  $V_1$  and  $V_2$ .

**Protocol  $\mathcal{P}_{\text{ZHLOCC}}$ :** Let  $R_{1,1}, \dots, R_{N,1}, R_{N+1,1}$  be  $(N+1)$  one-qubit registers from the prover to  $V_1$ , and  $R_{1,2}, \dots, R_{N,2}, R_{N+1,2}$  be  $(N+1)$  one-qubit registers from the prover to  $V_2$ , respectively.

1.  $V_1$  chooses a random permutation  $\pi$  on  $\{1, 2, \dots, N+1\}$  and sends it to  $V_2$ , and then both perform  $\pi$  on the  $(N+1)$  two-qubit registers  $(R_{1,1}, R_{1,2}), \dots, (R_{N,1}, R_{N,2}), (R_{N+1,1}, R_{N+1,2})$ . Rename  $R_{j,1} := R_{\pi(j),1}$  and  $R_{j,2} := R_{\pi(j),2}$ .
2.  $V_1$  chooses  $N$  random numbers  $k_1, k_2, \dots, k_N \in \{1, 2, 3\}$  and sends them to  $V_2$ . For each  $j = 1, 2, \dots, N$ ,  $V_1$  and  $V_2$  implement one of the POVMs  $M_1, M_2, M_3$  on register  $(R_{j,1}, R_{j,2})$  jointly as follows.
  - when  $k_j = 1$ , they jointly implement  $M_1 = \{E_1, I - E_1\}$ ;  $V_1$  and  $V_2$  measure  $R_{j,1}$  and  $R_{j,2}$  in the  $Z$  basis  $\{|0\rangle, |1\rangle\}$ , respectively, and  $V_1$  sends the measurement value to  $V_2$ , who rejects iff it differs from the measurement value of  $V_2$ .

- when  $k_j = 2$ , they jointly implement  $M_2 = \{E_2, I - E_2\}$ ;  $V_1$  and  $V_2$  measure  $R_{j,1}$  and  $R_{j,2}$  in the  $X$  basis  $\{|+\rangle, |-\rangle\}$ , respectively, and  $V_1$  sends the measurement value to  $V_2$ , who rejects iff it differs from the measurement value of  $V_2$ .
  - when  $k_j = 3$ , they jointly implement  $M_3 = \{E_3, I - E_3\}$ ;  $V_1$  and  $V_2$  measure  $R_{j,1}$  and  $R_{j,2}$  in the  $Y$  basis  $\{|+\rangle, |-\rangle\}$ , respectively, and  $V_1$  sends the measurement value to  $V_2$ , who rejects iff it is same as the measurement value of  $V_2$ .
3. The test passes and  $V_1$  and  $V_2$  output  $R_{N+1,1}$  and  $R_{N+1,2}$ , respectively.

It is easy to see that  $\mathcal{P}_{\text{ZHLOCC}}$  simulates  $\mathcal{P}_{\text{ZH}}$  exactly in a distributed manner. The protocol  $\mathcal{P}_{\text{ZHLOCC}}$  does not use any quantum communication between  $V_1$  and  $V_2$ , while the amount of classical communication used between  $V_1$  and  $V_2$  is  $\lceil \log(N+1)! \rceil + \lceil \log 3^N \rceil + N = O(N \log N)$ .

Furthermore, we can replace a random permutation  $\pi$  in step 1 of  $\mathcal{P}_{\text{ZHLOCC}}$  by switching the  $j$ th two-qubit register  $(R_{j,1}, R_{j,2})$  and the  $(N+1)$ th register  $(R_{N+1,1}, R_{N+1,2})$  by choosing  $j$  uniformly at random from  $\{1, 2, \dots, N+1\}$  (actually, doing nothing when  $j = N+1$ ) since the output state by such change is the same as protocol  $\mathcal{P}_{\text{ZHLOCC}}$ . We call the protocol by such change  $\mathcal{P}_{\text{ZHLOCC}}^+$ . Now the amount of classical communication used between  $V_1$  and  $V_2$  in  $\mathcal{P}_{\text{ZHLOCC}}^+$  is improved to  $\lceil \log(N+1) \rceil + \lceil \log 3^N \rceil + N = O(N)$ .

Thus the following theorem holds for  $\mathcal{P}_{\text{ZHLOCC}}^+$ .

► **Theorem 8.** *For the same number  $N = O(\frac{1}{\varepsilon} \log(\frac{1}{\delta}))$  as Theorem 7, if the test passed with at least probability  $\delta$ , then the two-qubit state  $\tilde{\sigma}$  output by  $V_1$  and  $V_2$  in  $\mathcal{P}_{\text{ZHLOCC}}^+$  satisfies  $\langle \Phi^+ | \tilde{\sigma} | \Phi^+ \rangle \geq 1 - \varepsilon$ .*

In the full version, we prove Theorem 5 by showing how to use the protocol  $\mathcal{P}_{\text{ZHLOCC}}^+$ .

---

## References

- 1 Scott Aaronson. The complexity of quantum states and transformations: from quantum money to black holes, 2016. [arXiv:1607.05256](https://arxiv.org/abs/1607.05256).
- 2 Joran van Apeldoorn and Tijn de Vos. A framework for distributed quantum queries in the CONGEST model. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC 2022)*, pages 109–119, 2022.
- 3 Heger Arfaoui and Pierre Fraigniaud. What can be computed without communications? *SIGACT News*, 45(3):82–104, 2014. doi:10.1145/2670418.2670440.
- 4 Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70:1895–1899, 1993.
- 5 Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87:167902, 2001. doi:10.1103/PhysRevLett.87.167902.
- 6 Matias Christandl, Robert König, Graeme Mitchison, and Renato Renner. One-and-a-half quantum de finetti theorem. *Communications in Mathematical Physics*, 273:473–498, 2007.
- 7 Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-offs in distributed interactive proofs. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC 2019)*, pages 13:1–13:17, 2019. doi:10.4230/LIPIcs.DISC.2019.13.
- 8 Vasil S. Denchev and Gopal Pandurangan. Distributed quantum computing: a new frontier in distributed systems or science fiction? *SIGACT News*, 39(3):77–95, 2008. doi:10.1145/1412700.1412718.
- 9 Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC 2014)*, pages 166–175, 2014.

- 10 Pierre Fraigniaud, François Le Gall, Harumichi Nishimura, and Ami Paz. Distributed quantum proofs for replicated data. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, pages 28:1–28:20, 2021.
- 11 Pierre Fraigniaud, Pedro Montealegre, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. On distributed Merlin-Arthur decision protocols. In *Proceedings of the 26th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2019)*, pages 230–245, 2019.
- 12 Pierre Fraigniaud, Boaz Patt-Shamir, and Mor Perry. Randomized proof-labeling schemes. *Distributed Computing*, 32(3):217–234, 2019. doi:10.1007/s00446-018-0340-8.
- 13 Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. What can be observed locally? In *Proceedings of the 23rd International Symposium on Distributed Computing (DISC 2009)*, pages 243–257, 2009.
- 14 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12:19:1–19:33, 2016. doi:10.4086/toc.2016.v012a019.
- 15 Masahito Hayashi and Tomoyuki Morimae. Verifiable measurement-only blind quantum computing with stabilizer testing. *Physical Review Letters*, 115:220502, 2015.
- 16 Sandy Irani, Anand Natarajan, Chinmay Nirkhe, Sujit Rao, and Henry Yuen. Quantum search-to-decision reductions and the state synthesis problem. In *Proceedings of the 37th Computational Complexity Conference (CCC 2022)*, pages 5:1–5:19, 2022.
- 17 Taisuke Izumi and François Le Gall. Quantum distributed algorithm for the All-Pairs Shortest Path problem in the CONGEST-CLIQUE model. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 84–93, 2019.
- 18 Taisuke Izumi, François Le Gall, and Frédéric Magniez. Quantum distributed algorithm for triangle finding in the CONGEST model. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, pages 23:1–23:13, 2020.
- 19 Benjamin Jauregui, Pedro Montealegre, and Ivan Rapaport. Distributed interactive proofs for the recognition of some geometric intersection graph classes. In *Proceedings of 29th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2022)*, pages 212–233, 2022.
- 20 Zhengfeng Ji, Yi-Kai Liu, and Fang Song. Pseudorandom quantum states. In *Proceedings of the 38th Annual International Cryptology Conference (CRYPTO 2018), Part III*, pages 126–152, 2018.
- 21 Martin Kliesch and Ingo Roth. Theory of quantum system certification. *PRX quantum*, 2(1):010201, 2021.
- 22 Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 255–264, 2018.
- 23 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.
- 24 William Kretschmer. Quantum pseudorandomness and classical complexity. In *Proceedings of the 16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*, pages 2:1–2:20, 2021.
- 25 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in CONGEST networks. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 337–346, 2018.
- 26 François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura. Distributed quantum interactive proofs. In *Proceedings of the 40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, pages 42:1–42:21, 2023.
- 27 François Le Gall, Harumichi Nishimura, and Ansis Rosmanis. Quantum advantage for the LOCAL model in distributed computing. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, pages 49:1–49:14, 2019.

- 28 Ke Li and Graeme Smith. Quantum de Finetti theorem under fully-one-way adaptive measurements. *Physical Review Letters*, 114:160503, 2015.
- 29 Tony Metger and Henry Yuen. stateQIP = statePSPACE. [arXiv:2301.07730](https://arxiv.org/abs/2301.07730). Presented at the 26th Conference on Quantum Information Processing (QIP2023).
- 30 Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Shared vs private randomness in distributed interactive proofs. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, pages 51:1–51:13, 2020.
- 31 Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Compact distributed interactive proofs for the recognition of cographs and distance-hereditary graphs. In *Proceedings of the International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS 2021)*, pages 395–409, 2021.
- 32 Tomoyuki Morimae, Yuki Takeuchi, and Masahito Hayashi. Verification of hypergraph states. *Physical Review A*, 96:062321, 2017.
- 33 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1096–115, 2020. doi:10.1137/1.9781611975994.67.
- 34 Sam Pallister, Noah Linden, and Ashley Montanaro. Optimal verification of entangled states with local measurements. *Physical Review Letters*, 120:170502, 2018. doi:10.1103/PhysRevLett.120.170502.
- 35 Gregory Rosenthal and Henry Yuen. Interactive proofs for synthesizing quantum states and unitaries. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, pages 112:1–112:4, 2022.
- 36 Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Exact quantum algorithms for the leader election problem. *ACM Transactions on Computation Theory*, 4(1):1:1–1:24, 2012. doi:10.1145/2141938.2141939.
- 37 Xudong Wu and Penghui Yao. Quantum complexity of weighted diameter and radius in CONGEST networks. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC 2022)*, pages 120–130, 2022.
- 38 Xiao-Dong Yu, Jiangwei Shang, and Otfried Gühne. Statistical methods for quantum state verification and fidelity estimation. *Advanced Quantum Technologies*, 5(5):2100126, 2022.
- 39 Huangjun Zhu and Masahito Hayashi. Efficient verification of pure quantum states in the adversarial scenario. *Physical Review Letters*, 123:260504, 2019.
- 40 Huangjun Zhu and Masahito Hayashi. General framework for verifying pure quantum states in the adversarial scenario. *Physical Review A*, 100:062335, 2019.
- 41 Huangjun Zhu and Masahito Hayashi. Optimal verification and fidelity estimation of maximally entangled states. *Physical Review A*, 99:052346, 2019.



# Dichotomies for Maximum Matching Cut: $H$ -Freeness, Bounded Diameter, Bounded Radius

Felicia Lucke  

Department of Informatics, University of Fribourg, Switzerland

Daniël Paulusma  

Department of Computer Science, Durham University, UK

Bernard Ries  

Department of Informatics, University of Fribourg, Switzerland

---

## Abstract

The (PERFECT) MATCHING CUT problem is to decide if a graph  $G$  has a (perfect) matching cut, i.e., a (perfect) matching that is also an edge cut of  $G$ . Both MATCHING CUT and PERFECT MATCHING CUT are known to be NP-complete, leading to many complexity results for both problems on special graph classes. A perfect matching cut is also a matching cut with maximum number of edges. To increase our understanding of the relationship between the two problems, we introduce the MAXIMUM MATCHING CUT problem. This problem is to determine a largest matching cut in a graph. We generalize and unify known polynomial-time algorithms for MATCHING CUT and PERFECT MATCHING CUT restricted to graphs of diameter at most 2 and to  $(P_6 + sP_2)$ -free graphs. We also show that the complexity of MAXIMUM MATCHING CUT differs from the complexities of MATCHING CUT and PERFECT MATCHING CUT by proving NP-hardness of MAXIMUM MATCHING CUT for  $2P_3$ -free quadrangulated graphs of diameter 3 and radius 2 and for subcubic line graphs of triangle-free graphs. In this way, we obtain full dichotomies of MAXIMUM MATCHING CUT for graphs of bounded diameter, bounded radius and  $H$ -free graphs.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Graph algorithms

**Keywords and phrases** matching cut, perfect matching,  $H$ -free graph, diameter, radius, dichotomy

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.64

## 1 Introduction

A *matching*  $M$  (i.e., a set of pairwise disjoint edges) of a connected graph  $G = (V, E)$  is a *matching cut* if  $V$  can be partitioned into a set of blue vertices  $B$  and a set of red vertices  $R$ , such that  $M$  consists of all the edges with one end-vertex in  $B$  and the other one in  $R$ . Graphs with matching cuts were introduced in 1970 by Graham [20] (as *decomposable* graphs) to solve a problem on cube numbering. Other relevant applications include ILFI networks [13], WDM networks [1], graph drawing [33] and surjective graph homomorphisms [18].

The decision problem is called MATCHING CUT: does a given connected graph have a matching cut? In 1984, Chvátal [11] proved that it is NP-complete even for graphs of maximum degree at most 4. Afterwards, parameterized and exact algorithms were given [2, 8, 17, 19, 24, 25]. A variant called DISCONNECTED PERFECT MATCHING “does a connected graph have a perfect matching that contains a matching cut?” has also been studied [7, 15, 31], and the problem was generalized, for every  $d \geq 1$ , to  $d$ -CUT “does a connected graph have an edge cut where each vertex has at most  $d$  neighbours across the cut?” [3, 19]. But, in particular, many results have appeared where the input for MATCHING CUT was restricted to some special graph class, and this is what we do in our paper as well. We first discuss related work, restricting ourselves mainly to those classes relevant to our paper (see, for example, [8] for a more comprehensive overview):



© Felicia Lucke, Daniël Paulusma, and Bernard Ries;  
licensed under Creative Commons License CC-BY 4.0

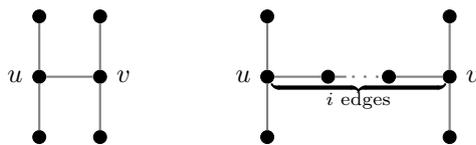
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 64; pp. 64:1–64:15

Leibniz International Proceedings in Informatics



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



■ **Figure 1** The graphs  $H_1^*$  (left) and  $H_i^*$  (right).

- graphs of bounded diameter;
- graphs of bounded radius;
- hereditary graph classes; in particular  $H$ -free graphs.

The *distance* between two vertices  $u$  and  $v$  in a connected graph  $G$  is the *length* (number of edges) of a shortest path between  $u$  and  $v$  in  $G$ . The *eccentricity* of a vertex  $u$  is the maximum distance between  $u$  and any other vertex of  $G$ . The *diameter*, denoted by  $\text{diameter}(G)$ , and *radius*, denoted by  $\text{radius}(G)$ , are the maximum and minimum eccentricity, respectively, over all vertices of  $G$ ; note that  $\text{radius}(G) \leq \text{diameter}(G) \leq 2 \cdot \text{radius}(G)$  for every graph  $G$ .

The MATCHING CUT problem is polynomial-time solvable for graphs of diameter at most 2 [6, 26]. This result was extended to graphs of radius at most 2 [30]. In contrast, the problem is NP-complete for graphs of diameter at most 3 [26], yielding two dichotomies:

► **Theorem 1** ([26, 30]). *For an integer  $d \geq 1$ , MATCHING CUT for graphs of diameter  $d$  and for graphs of radius  $d$  is polynomial-time solvable if  $d \leq 2$  and NP-complete if  $d \geq 3$ .*

A class of graphs is *hereditary* if it is closed under vertex deletion. Hereditary graph classes include many well-known classes, such as those that are  $H$ -free for some graph  $H$ . A graph  $G$  is  $H$ -free if  $G$  does not contain  $H$  as an *induced* subgraph, that is,  $G$  cannot be modified into  $H$  by a sequence of vertex deletions. For a set of graphs  $\mathcal{H}$ , a graph  $G$  is  $\mathcal{H}$ -free if  $G$  is  $H$ -free for every  $H \in \mathcal{H}$ . If  $\mathcal{H} = \{H_1, \dots, H_p\}$  for some  $p \geq 1$ , we also say that  $G$  is  $(H_1, \dots, H_p)$ -free. Note that a class of graphs  $\mathcal{G}$  is hereditary if and only if there is a set of graphs  $\mathcal{H}$ , such that every graph in  $\mathcal{G}$  is  $\mathcal{H}$ -free. Hence, for a *systematic* complexity study, it is natural to first focus on the case where  $\mathcal{H}$  has size 1; see, e.g., [9, 10, 12, 16, 22, 35].

For an integer  $r \geq 1$ , let  $P_r$  denote the path on  $r$  vertices,  $K_{1,r}$  the star on  $r + 1$  vertices, and  $K_{1,r} + e$  the graph obtained from  $K_{1,r}$  by adding one edge (between two leaves). The graph  $K_{1,3}$  is also known as the *claw*. For  $s \geq 3$ , let  $C_s$  denote the cycle on  $s$  vertices. Let  $H_1^*$  be the graph that looks like the letter “H”, and for  $i \geq 2$ , let  $H_i^*$  be the graph obtained from  $H_1^*$  by subdividing the middle edge of  $H_1^*$  exactly  $i - 1$  times; see also Figure 1. We denote the *disjoint union* of two graphs  $G_1$  and  $G_2$  by  $G_1 + G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$ . We denote by  $sG$  the disjoint union of  $s$  copies of  $G$ , for  $s \geq 1$ .

Polynomial-time algorithms for MATCHING CUT exist for *subcubic* graphs (graphs of maximum degree at most 3) [11],  $K_{1,3}$ -free graphs [5],  $P_6$ -free graphs [30],  $(K_{1,4}, K_{1,4} + e)$ -free graphs [25] and *quadrangulated* graphs, i.e.,  $(C_5, C_6, \dots)$ -free graphs [32]; the latter class contains the class of *chordal* graphs, i.e.,  $(C_4, C_5, C_6, \dots)$ -free graphs. Moreover, if MATCHING CUT is polynomial-time solvable for  $H$ -free graphs, then it is so for  $(H + P_3)$ -free graphs [30]. The problem is NP-complete even for graphs of maximum degree at most 4 [11];  $K_{1,4}$ -free graphs [11] (see [5, 25]); planar graphs of girth 5 [5];  $K_{1,5}$ -free bipartite graphs [28]; graphs of girth at least  $g$ , for every  $g \geq 3$  [15];  $(3P_5, P_{15})$ -free graphs [31] (improving a result of [14]); bipartite graphs where the vertices in one bipartition class all have degree exactly 2 [32] and thus for  $H_i^*$ -free graphs for every odd  $i \geq 1$ ; and for  $H_i^*$ -free graphs for every even  $i \geq 2$  [15]. Even more recently, Le and Le [27] proved that MATCHING CUT is NP-complete even for  $(3P_6, 2P_7, P_{14})$ -free graphs.

The above results imply the following partial complexity classification, which leaves open only a number of cases where  $H$  is a *linear forest*, that is, the disjoint union of one or more paths. For two graphs  $H$  and  $H'$ , we write  $H \subseteq_i H'$  if  $H$  is an induced subgraph of  $H'$ .

► **Theorem 2** ([5, 11, 15, 27, 31, 30, 32]). *For a graph  $H$ , MATCHING CUT on  $H$ -free graphs is*

- *polynomial-time solvable if  $H \subseteq_i sP_3 + K_{1,3}$  or  $sP_3 + P_6$  for some  $s \geq 0$ , and*
- *NP-complete if  $H \supseteq_i K_{1,4}, P_{14}, 3P_5, 2P_7, C_r$  for some  $r \geq 3$  or  $H_j^*$  for some  $j \geq 1$ .*



■ **Figure 2** The graph  $P_6$  with a matching cut of size 2 (left), another matching cut of size 2 (middle) and a perfect matching cut (right). In each figure, thick edges denote matching cut edges.

### 1.1 Our Focus

We already mentioned the known generalization of MATCHING CUT (i.e. 1-CUT) to  $d$ -CUT. In our paper, we consider a different kind of generalization, namely MAXIMUM MATCHING CUT, which is to determine a *largest* matching cut of a connected graph (if a matching cut exists). So far, this problem has only been studied for the extreme case, where the task is to decide if a connected graph has a *perfect* matching cut, that is, a matching cut that saturates every vertex; see also Figure 2. This variant was introduced as PERFECT MATCHING CUT by Heggenes and Telle [21], who proved it was NP-complete. We briefly discuss some very recent results for PERFECT MATCHING CUT on special graph classes below.

It is readily seen that the gadget in the NP-hardness reduction of Heggenes and Telle [21] has diameter 6 and radius 3. More recently Le and Le [27] gave a reduction with a graph of diameter 4. It is also known that PERFECT MATCHING CUT is polynomial-time solvable for graphs of radius (and thus also diameter) at most 2 [31]. Hence, we only obtain a partial complexity classification for graphs of bounded diameter in this case.

► **Theorem 3** ([21, 31]). *For integers  $d$  and  $r$ , PERFECT MATCHING CUT for graphs of diameter  $d$  and for graphs of radius  $r$  is polynomial-time solvable if  $d \leq 2$  or  $r \leq 2$ , respectively, and NP-complete if  $d \geq 4$  or  $r \geq 3$ , respectively.*

For  $1 \leq h \leq i \leq j$ , the graph  $S_{h,i,j}$  is the tree of maximum degree 3 with exactly one vertex  $u$  of degree 3, whose leaves are at distance  $h, i$  and  $j$ , respectively, from  $u$ ; note  $S_{1,1,1} = K_{1,3}$ .

It is known that PERFECT MATCHING CUT is polynomial-time solvable for  $S_{1,2,2}$ -free graphs (and thus for  $K_{1,3}$ -free graphs) [29];  $P_6$ -free graphs [31]; and for pseudo-chordal graphs [29] (and thus for chordal graphs, i.e.,  $(C_4, C_5, \dots)$ -free graphs). Moreover, PERFECT MATCHING CUT is polynomial-time solvable for  $(H + P_4)$ -free graphs if it is polynomial-time solvable for  $H$ -free graphs [31]. It is also known that PERFECT MATCHING CUT is NP-complete even for 3-connected cubic planar bipartite graphs [4],  $(3P_6, 2P_7, P_{14})$ -free graphs [27],  $K_{1,4}$ -free bipartite graphs of girth  $g$  for every  $g \geq 3$  [29] and for  $H_i^*$ -free graphs for every  $i \geq 1$  [15]. This gives us a partial complexity classification:

► **Theorem 4** ([15, 27, 29, 31]). *For a graph  $H$ , PERFECT MATCHING CUT on  $H$ -free graphs is*

- *polynomial-time solvable if  $H \subseteq_i sP_4 + S_{1,2,2}$  or  $sP_4 + P_6$  for some  $s \geq 0$ , and*
- *NP-complete if  $H \supseteq_i K_{1,4}, P_{14}, 3P_6, 2P_7, C_r$  for some  $r \geq 3$  or  $H_j^*$  for some  $j \geq 1$ .*

From Theorem 4 it can be seen that again only cases where  $H$  is a linear forest remain open. However, the number of open cases is smaller than for MATCHING CUT. So far, all known complexities for MATCHING CUT and PERFECT MATCHING CUT on special graph classes coincide except for (sub)cubic graphs. We note that whenever MAXIMUM MATCHING CUT is polynomial-time solvable for some graph class, then so are MATCHING CUT and PERFECT MATCHING CUT. Similarly, if one of the latter two problems is NP-complete, then MAXIMUM MATCHING CUT is NP-hard. For instance, this immediately yields a complexity dichotomy for graphs of maximum degree at most  $\Delta$ . Namely, as MAXIMUM MATCHING CUT is trivial if  $\Delta = 2$  and PERFECT MATCHING CUT is NP-complete if  $\Delta = 3$ , we have a complexity jump from  $\Delta = 2$  to  $\Delta = 3$ , just like PERFECT MATCHING CUT; recall that for MATCHING CUT this jump appears from  $\Delta = 3$  to  $\Delta = 4$ . We consider the following research question:

*For which graph classes is MAXIMUM MATCHING CUT harder than MATCHING CUT and PERFECT MATCHING CUT and for which graph classes do the complexities coincide?*

## 1.2 Our Results

In Section 4 we show that MAXIMUM MATCHING CUT is NP-hard for  $2P_3$ -free quadrangulated graphs of diameter 3 and radius 2. We note that the restrictions to radius 2 and diameter 3 are not redundant: consider, for example, the  $P_6$ , which is  $2P_3$ -free but which has radius 3 and diameter 5. In the same section, we also show NP-hardness for subcubic line graphs of triangle-free graphs, or equivalently, subcubic  $(K_{1,3}, \text{diamond})$ -free graphs (the diamond is obtained from the  $K_4$  after removing an edge). These NP-hardness results are in stark contrast to the situation for MATCHING CUT and PERFECT MATCHING CUT, as evidenced by Theorems 1–4. Recall also that MATCHING CUT is polynomial-time solvable for quadrangulated graphs [32].

Before proving these results, we first show in Section 3 that MAXIMUM MATCHING CUT is polynomial-time solvable for graphs of diameter 2, generalizing the known polynomial-time algorithms for MATCHING CUT and PERFECT MATCHING CUT for graphs of diameter at most 2. Hence, all three problems have the same dichotomies for graphs of bounded diameter.

We also prove in Section 3 that MAXIMUM MATCHING CUT is polynomial-time solvable for  $P_6$ -free graphs, generalizing the previous polynomial-time results for MATCHING CUT and PERFECT MATCHING CUT for  $P_6$ -free graphs. Due to the hardness result for  $2P_3$ -free graphs, we cannot show polynomial-time solvability for “ $+P_4$ ” (as for PERFECT MATCHING CUT) or “ $+P_3$ ” (as for MATCHING CUT). However, we can prove that if MAXIMUM MATCHING CUT is polynomial-time solvable for  $H$ -free graphs, then it is so for  $(H + P_2)$ -free graphs; again, see Section 3. The common proof technique for our polynomial-time results is as follows:

1. Translate the problem into a colouring problem. We pre-colour some vertices either red or blue, and try to extend the pre-colouring to a red-blue colouring of the whole graph via reduction rules. This technique has been used for MATCHING CUT and PERFECT MATCHING CUT, but our analysis is different. In particular, the algorithms for MATCHING CUT and PERFECT MATCHING CUT on  $P_6$ -free graphs use an algorithm for graphs of radius at most 2 as a subroutine (shortcut). We cannot do this for MAXIMUM MATCHING CUT, as we will show NP-hardness for radius 2.
2. Reduce the set of uncoloured vertices, via a number of branching steps, to an independent set, and then translate the problem into a matching problem. This is a new proof ingredient. The matching problem is to find a largest matching that saturates every vertex of the independent set of uncoloured vertices. Plesník [34] gave a polynomial time algorithm for this, which we will use as subroutine.<sup>1</sup>

---

<sup>1</sup> The polynomial-time algorithm of Plesník [34] solves a more general problem. It takes as input a

The above polynomial-time and NP-hardness results yield the following three dichotomies for MAXIMUM MATCHING CUT proven in Section 5; in particular we have obtained a complete complexity classification of MAXIMUM MATCHING CUT for  $H$ -free graphs (whereas such a classification is only partial for the other two problems, as shown in Theorems 2 and 4).

► **Theorem 5.** *For an integer  $d$ , MAXIMUM MATCHING CUT on graphs of diameter  $d$  is*

- polynomial-time solvable if  $d \leq 2$ , and
- NP-hard if  $d \geq 3$ .

► **Theorem 6.** *For an integer  $r$ , MAXIMUM MATCHING CUT on graphs of radius  $r$  is*

- polynomial-time solvable if  $r \leq 1$ , and
- NP-hard if  $r \geq 2$ .

► **Theorem 7.** *For a graph  $H$ , MAXIMUM MATCHING CUT on  $H$ -free graphs is*

- polynomial-time solvable if  $H \subseteq_i sP_2 + P_6$  for some  $s \geq 0$ , and
- NP-hard if  $H \supseteq_i K_{1,3}$ ,  $2P_3$  or  $H \supseteq_i C_r$  for some  $r \geq 3$ .

Finally, in Section 6 we pose a number of open problems.

## 2 Preliminaries

We consider finite, undirected graphs without multiple edges and self-loops. Let  $G = (V, E)$  be a connected graph. For  $u \in V$ , the set  $N(u) = \{v \in V(G) \mid uv \in E(G)\}$  is the *neighbourhood* of  $u$  in  $G$ , where  $|N(u)|$  is the *degree* of  $u$ . For  $S \subseteq V$ , the *neighbourhood* of  $S$  is the set  $N(S) = \bigcup_{u \in S} N(u) \setminus S$ . The graph  $G[S]$  is the subgraph of  $G$  induced by  $S \subseteq V(G)$ , that is,  $G[S]$  is the graph obtained from  $G$  after deleting the vertices not in  $S$ . We say that  $S$  is a *dominating* set of  $G$ , and that  $G[S]$  *dominates*  $G$  if every vertex of  $V(G) \setminus S$  has at least one neighbour in  $S$ . The *domination number* of  $G$  is the size of a smallest dominating set of  $G$ . The set  $S$  is an *independent set* if no two vertices in  $S$  are adjacent and  $S$  is a *clique* if every two vertices in  $S$  are adjacent. A matching  $M$  is  *$S$ -saturating* if every vertex in  $S$  is an end-vertex of an edge in  $M$ . An  $S$ -saturating matching is *maximum* if there is no  $S$ -saturating matching of  $G$  with more edges. We will use the following result.

► **Theorem 8** ([34]). *For a graph  $G$  and set  $S \subseteq V(G)$ , it is possible in polynomial time to find a maximum  $S$ -saturating matching or conclude that  $G$  has no  $S$ -saturating matching.*

The *line graph* of  $G$  is the graph  $L(G)$  whose vertices are the edges of  $G$ , such that for every two vertices  $e$  and  $f$ , there exists an edge between  $e$  and  $f$  in  $L(G)$  if and only if  $e$  and  $f$  share an end-vertex in  $G$ . A bipartite graph with non-empty partition classes  $V_1$  and  $V_2$  is *complete* if there is an edge between every vertex of  $V_1$  and every vertex of  $V_2$ . If  $|V_1| = k$  and  $|V_2| = \ell$ , then we denote the complete bipartite graph by  $K_{k,\ell}$ . We will need the following theorem.

► **Theorem 9** ([36]). *A graph  $G$  on  $n$  vertices is  $P_6$ -free if and only if each connected induced subgraph of  $G$  contains a dominating induced  $C_6$  or a dominating (not necessarily induced) complete bipartite graph. We can find such a dominating subgraph of  $G$  in  $O(n^3)$  time.*

A *red-blue colouring* of a connected graph  $G$  colours every vertex of  $G$  either red or blue. If every vertex of a set  $S \subseteq V$  has the same colour (red or blue), then  $S$ , and also  $G[S]$ , are called *monochromatic*. An edge with a blue and a red end-vertex is *bichromatic*. A red-blue

---

graph  $G$  with an edge weighting  $w$ , a vertex subset  $S$  and two integers  $a$  and  $b$ . It then finds a maximum weight matching over all matchings that saturate  $S$  and whose cardinality is between  $a$  and  $b$ .

colouring is *valid* if every blue vertex has at most one red neighbour; every red vertex has at most one blue neighbour; and both colours red and blue are used at least once. For a valid red-blue colouring of  $G$ , we let  $R$  be the *red* set consisting of all vertices coloured red and  $B$  be the *blue* set consisting of all vertices coloured blue (so  $V(G) = R \cup B$ ). Moreover, the *red interface* is the set  $R' \subseteq R$  consisting of all vertices in  $R$  with a (unique) blue neighbour, and the *blue interface* is the set  $B' \subseteq B$  consisting of all vertices in  $B$  with a (unique) red neighbour in  $R$ . The *value* of a valid red-blue colouring is its number of bichromatic edges, or equivalently, the size of its red (or blue) interface. A valid red-blue colouring is *maximum* if there is no valid red-blue colouring of the graph with larger value. The notion of a red-blue colouring has been used before (see e.g. [14, 30]), and the next observations are easy to see.

► **Observation 10.** *For every connected graph  $G$  and integer  $k$ , it holds that  $G$  has a matching cut with at least  $k$  edges if and only if  $G$  has a valid red-blue colouring of value at least  $k$ .*

► **Observation 11.** *Every complete graph  $K_r$  with  $r \geq 3$  and every complete bipartite graph  $K_{r,s}$  with  $\min\{r, s\} \geq 2$  and  $\max\{r, s\} \geq 3$  is monochromatic.*

We omitted the proof of our next lemma; it is very similar to the proofs of corresponding lemmas for MATCHING CUT [14] and PERFECT MATCHING CUT [31]. On an aside, the lemma implies that MAXIMUM MATCHING CUT is in XP when parameterized by the domination number of a graph.

► **Lemma 12.** *For a connected  $n$ -vertex graph  $G$  with domination number  $g$ , it is possible to find a maximum red-blue colouring (if a red-blue colouring exists) in  $O(2^g n^{g+2})$  time.*

To handle “partial” red-blue colourings that we want to extend to maximum valid red-blue colourings, we slightly modify some terminology from [31] to work for maximum matching cuts as well.

Let  $G = (V, E)$  be a connected graph and  $S, T, X, Y \subseteq V$  be four non-empty sets with  $S \subseteq X$ ,  $T \subseteq Y$  and  $X \cap Y = \emptyset$ . A *red-blue  $(S, T, X, Y)$ -colouring* of  $G$  is a red-blue colouring of the vertices in  $X \cup Y$ , with a red set containing  $X$ ; a blue set containing  $Y$ ; a red interface containing  $S$  and a blue interface containing  $T$ . To obtain a red-blue  $(S, T, X, Y)$ -colouring, we start with two disjoint subsets  $S''$  and  $T''$  of  $V$ , called a *starting pair*, such that

- (i) every vertex of  $S''$  is adjacent to at most one vertex of  $T''$ , and vice versa, and
- (ii) at least one vertex in  $S''$  is adjacent to a vertex in  $T''$ .

Let  $S^*$  consist of all vertices of  $S''$  with a (unique) neighbour in  $T''$ , and let  $T^*$  consist of all vertices of  $T''$  with a (unique) neighbour in  $S''$ ; so, every vertex in  $S^*$  has a unique neighbour in  $T^*$ , and vice versa. We call  $(S^*, T^*)$  the *core* of  $(S'', T'')$ . Note that  $|S^*| = |T^*| \geq 1$ .

We now colour every vertex in  $S''$  red and every vertex in  $T''$  blue. Propagation rules will try to extend  $S''$  to a set  $X$ , and  $T''$  to a set  $Y$ , by finding new vertices whose colour must always be either red or blue. That is, we place new red vertices in the set  $X$ , which already contains  $S''$ , and new blue vertices in the set  $Y$ , which already contains  $T''$ . If a red and blue vertex are adjacent, then we add the red one to a set  $S \subseteq X$  and the blue one to a set  $T \subseteq Y$ . So initially,  $S := S^*$ ,  $T := T^*$ ,  $X := S''$  and  $Y := T''$ . We let  $Z := V \setminus (X \cup Y)$ .

Our task is to try to extend the partial red-blue colouring on  $X \cup Y$  to a *maximum* valid red-blue  $(S, T, X, Y)$ -colouring of  $G$ , that is, a valid red-blue  $(S, T, X, Y)$ -colouring that has largest value over all valid red-blue  $(S, T, X, Y)$ -colourings of  $G$ . In order to do this, we present three propagation rules, which indicate necessary implications of previous choices.

We start with rules R1 and R2, which together correspond to the five rules from [26]. Rule R1 detects cases where we cannot extend the partial red-blue colouring defined on  $X \cup Y$ . Rule R2 tries to extend the sets  $S, T, X, Y$  as much as possible. While the sets  $S, T, X, Y$  grow, Rule R2 ensures that we keep constructing a (maximum) valid red-blue  $(S, T, X, Y)$ -colouring (assuming  $G$  has a valid red-blue  $(S, T, X, Y)$ -colouring).

- R1. Return no (i.e.,  $G$  has no red-blue  $(S, T, X, Y)$ -colouring) if a vertex  $v \in Z$  is
- (i) adjacent to a vertex in  $S$  and to a vertex in  $T$ , or
  - (ii) adjacent to a vertex in  $S$  and to two vertices in  $Y \setminus T$ , or
  - (iii) adjacent to a vertex in  $T$  and to two vertices in  $X \setminus S$ , or
  - (iv) adjacent to two vertices in  $X \setminus S$  and to two vertices in  $Y \setminus T$ .
- R2. Let  $v \in Z$ .
- (i) If  $v$  is adjacent to a vertex in  $S$  or to two vertices of  $X \setminus S$ , then move  $v$  from  $Z$  to  $X$ . If  $v$  is also adjacent to a vertex  $w$  in  $Y$ , then add  $v$  to  $S$  and  $w$  to  $T$ .
  - (ii) If  $v$  is adjacent to a vertex in  $T$  or to two vertices of  $Y \setminus T$ , then move  $v$  from  $Z$  to  $Y$ . If  $v$  is also adjacent to a vertex  $w$  in  $X$ , then add  $v$  to  $T$  and  $w$  to  $S$ .

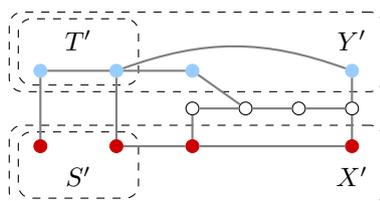
Assume that exhaustively applying rules R1 and R2 on a starting pair  $(S'', T'')$  does not lead to a no-answer but to a tuple  $(S', T', X', Y')$ . Then, we call  $(S', T', X', Y')$  an *intermediate tuple*; see also Figure 3. A propagation rule is *safe* if for every integer  $\nu$  the following holds:  $G$  has a valid red-blue  $(S, T, X, Y)$ -colouring of value  $\nu$  before the application of the rule if and only if  $G$  has a valid red-blue  $(S, T, X, Y)$ -colouring of value  $\nu$  after the application of the rule. Le and Le [26] proved the following lemma, which shows that R1 and R2 can be used safely and which is not difficult to verify. The fact that the value  $\nu$  is preserved in Lemma 13 (ii) below is implicit in their proof.

► **Lemma 13** ([26]). *Let  $G$  be a connected graph with a starting pair  $(S'', T'')$  with core  $(S^*, T^*)$ , and with an intermediate tuple  $(S', T', X', Y')$ . The following holds:*

- (i)  $S^* \subseteq S'$ ,  $S'' \subseteq X'$  and  $T^* \subseteq T'$ ,  $T'' \subseteq Y'$  and  $X' \cap Y' = \emptyset$ ,
- (ii) *For every integer  $\nu$ ,  $G$  has a valid red-blue  $(S^*, T^*, S'', T'')$ -colouring of value  $\nu$  if and only if  $G$  has a valid red-blue  $(S', T', X', Y')$ -colouring of value  $\nu$  (note that the backward implication holds by definition), and*
- (iii) *every vertex in  $S'$  (resp.  $T'$ ) has exactly one neighbour in  $Y'$  (resp. in  $X'$ ), which belongs to  $T'$  (resp.  $S'$ ); every vertex in  $X' \setminus S'$  (resp.  $Y' \setminus T'$ ) has no neighbour in  $Y'$  (resp.  $X'$ ); and every vertex of  $V \setminus (X' \cup Y')$  has no neighbour in  $S' \cup T'$ , at most one neighbour in  $X' \setminus S'$ , and at most one neighbour in  $Y' \setminus T'$ .*

Moreover,  $(S', T', X', Y')$  is obtained in polynomial time.

Let  $(S', T', X', Y')$  be an intermediate tuple of a graph  $G$ . Let  $Z = V \setminus (X' \cup Y')$ . A red-blue  $(S', T', X', Y')$ -colouring of  $G$  is *monochromatic* if all connected components of  $G[Z]$  are monochromatic. We say that an intermediate tuple  $(S', T', X', Y')$  is *monochromatic* if every connected component of  $G[V \setminus (X' \cup Y')]$  is monochromatic in every valid red-blue  $(S', T', X', Y')$ -colouring of  $G$ . A propagation rule is *mono-safe* if for every integer  $\nu$  the following holds:  $G$  has a valid monochromatic red-blue  $(S, T, X, Y)$ -colouring of value  $\nu$  before the application of the rule if and only if  $G$  has a valid monochromatic red-blue  $(S, T, X, Y)$ -colouring of value  $\nu$  after the application of the rule.



■ **Figure 3** An example (from [31]) of a red-blue  $(S', T', X', Y')$ -colouring of a graph with an intermediate 4-tuple  $(S', T', X', Y')$ .

We now present Rule R3 (used implicitly in [26]) and prove that R3 is mono-safe.

**R3.** If there are two distinct vertices  $u$  and  $v$  in a connected component  $D$  of  $G[Z]$  with a common neighbour  $w \in X \cup Y$ , then colour every vertex of  $D$  with the colour of  $w$ .

► **Lemma 14.** *Rule R3 is mono-safe.*

**Proof.** Say  $w \in X \cup Y$  is in  $X$ , so  $w$  is red. Then, at least one of  $x$  and  $y$  must be coloured red. Hence, as  $D$  must be monochromatic, every vertex of  $D$  must be coloured red. Note that the value of a maximum monochromatic red-blue  $(S, T, X, Y)$ -colouring (if it exists) is not affected. ◀

Suppose that exhaustively applying rules R1–R3 on an intermediate tuple  $(S', T', X', Y')$  does not lead to a no-answer but to a tuple  $(S, T, X, Y)$ . We call  $(S, T, X, Y)$  the *final* tuple. The following lemma can be proved by a straightforward combination of the arguments of the proof of Lemma 13 with Lemma 14 and the observation that an application of R3 takes polynomial time, just as a check to see if R3 can be applied.

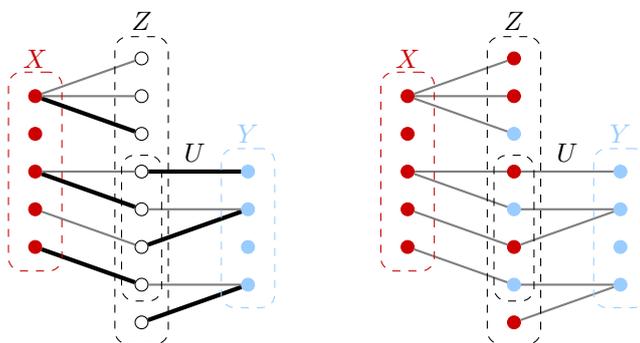
► **Lemma 15.** *Let  $G$  be a connected graph with a monochromatic intermediate tuple  $(S', T', X', Y')$  and a resulting final tuple  $(S, T, X, Y)$ . The following three statements hold:*

- (i)  $S' \subseteq S$ ,  $X' \subseteq X$ ,  $T' \subseteq T$ ,  $Y' \subseteq Y$ , and  $X \cap Y = \emptyset$ ,
- (ii) *For every integer  $\nu$ ,  $G$  has a valid (monochromatic) red-blue  $(S', T', X', Y')$ -colouring of value  $\nu$  if and only if  $G$  has a valid monochromatic red-blue  $(S, T, X, Y)$ -colouring of value  $\nu$  (note that the backward implication holds by definition), and*
- (iii) *every vertex in  $S$  (resp.  $T$ ) has exactly one neighbour in  $Y$  (resp.  $X$ ), which belongs to  $T$  (resp.  $S$ ); every vertex in  $X \setminus S$  (resp.  $Y \setminus T$ ) has no neighbour in  $Y$  (resp.  $X$ ) and no two neighbours in the same connected component of  $G[V \setminus (X \cup Y)]$ ; and every vertex of  $V \setminus (X \cup Y)$  has no neighbour in  $S \cup T$ , at most one neighbour in  $X \setminus S$ , and at most one neighbour in  $Y \setminus T$ .*

Moreover,  $(S, T, X, Y)$  is obtained in polynomial time.

### 3 Polynomial-Time Results

The following lemma uses Theorem 8 and is the final step in *all* our polynomial-time results.



■ **Figure 4** A  $U$ -saturating matching (left) and the corresponding valid red-blue colouring (right). Note that not every vertex in  $X \cup Y$  belongs to  $W$ .

► **Lemma 16.** *Let  $G = (V, E)$  be a connected graph with a monochromatic intermediate tuple  $(S', T', X', Y')$  and a final tuple  $(S, T, X, Y)$ . If  $V \setminus (X \cup Y)$  is an independent set, then it is possible to find in polynomial time either a maximum valid red-blue  $(S, T, X, Y)$ -colouring of  $G$  or conclude that  $G$  has no such colouring.*

**Proof.** Let  $Z = V \setminus (X \cup Y)$  and  $W = N(Z)$ . As  $Z$  is independent, every vertex of  $W$  belongs to  $(X \setminus S) \cup (Y \setminus T)$  due to Lemma 15-(iii). Let  $U \subseteq Z$  consist of all vertices of  $Z$  with a neighbour in both  $X \setminus S$  and  $Y \setminus T$ . We claim that the set of bichromatic edges of every valid red-blue  $(S, T, X, Y)$ -colouring is the union of a  $U$ -saturating matching in  $G[W \cup Z]$  (if it exists) and the set of edges with one end-vertex in  $S$  and the other one in  $T$ .

First suppose that  $G[W \cup Z]$  has a  $U$ -saturating matching  $M$ . We colour every vertex in  $X$  red and every vertex in  $Y$  blue. Let  $z \in Z$ . First assume that  $z$  is incident to an edge  $zw \in M$ . If  $w \in X \setminus S$ , then colour  $z$  blue. If  $w \in Y \setminus T$ , then colour  $z$  red. Now suppose  $z$  is not incident to an edge in  $M$ . Then  $z \notin U$ , as  $M$  is  $U$ -saturating. Hence, either every neighbour of  $z$  belongs to  $X \setminus S$  and is coloured red, in which case we colour  $z$  red, or every neighbour of  $z$  belongs to  $Y \setminus T$  and is coloured blue, in which case we colour  $z$  blue. This gives us a valid red-blue  $(S, T, X, Y)$ -colouring of  $G$ . See also Figure 4.

Now suppose that  $G$  has a valid red-blue  $(S, T, X, Y)$ -colouring. By definition, every vertex of  $X$  is coloured red, and every vertex of  $Y$  is coloured blue. By Lemma 15-(iii), every edge with an end-vertex in  $S$  and the other one in  $T$  is bichromatic, and there are no other bichromatic edges in  $G[X \cup Y]$ . Let  $M$  be the set of other bichromatic edges. Then, every vertex of  $M$  has one vertex in  $Z$  and the other one in  $W$ . Moreover, if  $z \in U$ , then  $z$  has a red neighbour (its neighbour in  $X \setminus S$ ) and a blue neighbour (its neighbour in  $Y \setminus T$ ). Hence, no matter what colour  $z$  has itself,  $z$  is incident to a bichromatic edge of  $M$ . We conclude that  $M$  is  $U$ -saturating, and the claim is proven.

From the above claim, it follows that all we have to do is to find a maximum  $U$ -saturating matching in  $G[W \cup Z]$ . By Theorem 8, this takes polynomial time. ◀

We are now ready to present our first result.

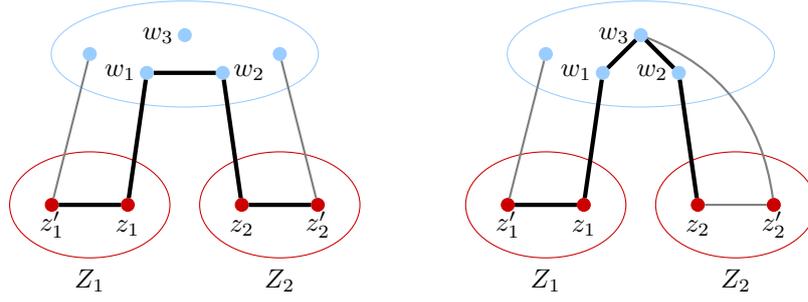
► **Theorem 17.** *MAXIMUM MATCHING CUT is solvable in polynomial time for  $P_6$ -free graphs.*

**Proof.** Let  $G = (V, E)$  be a connected  $P_6$ -free graph. By Observation 10 it suffices to find a maximum valid red-blue colouring of  $G$ . We know from Theorem 9 that  $G$  has a dominating induced  $C_6$  or a dominating (not necessarily induced) complete bipartite graph  $K_{r,s}$ , which can be found in polynomial time. If  $G$  has a dominating induced  $C_6$ , then  $G$  has domination number at most 6, and we apply Lemma 12. Suppose  $G$  has a dominating complete bipartite graph  $F$  with partition classes  $\{u_1, \dots, u_r\}$  and  $\{v_1, \dots, v_s\}$ , where  $r \leq s$ . If  $s \leq 2$ , then  $G$  has domination number at most 4, and we apply Lemma 12 again. So we assume that  $s \geq 3$ .

If  $r \geq 2$ , then  $V(F)$  must be monochromatic in any valid red-blue colouring of  $G$  by Observation 11. In this case we colour every vertex of  $V(F)$  blue. If  $r = 1$ , then we may assume without loss of generality that  $N(u_1) = \{v_1, \dots, v_s\}$ . In this case we colour  $u_1$  blue, and we branch over all  $O(n)$  options of colouring at most one vertex of  $N(u_1)$  red.

So, now we consider a red-blue colouring of  $F$ . It might be that  $F$  is monochromatic (in particular, this will be the case if  $r \geq 2$ ). If  $F$  is monochromatic, then every vertex of  $F$  is blue. In order to get a starting pair with a non-empty core, we branch over all  $O(n^2)$  options of choosing a bichromatic edge (one end-vertex of which may belong to  $F$ ). Let  $D$  be the set of all coloured vertices, that is,  $D$  contains  $V(F)$  and possibly one or two other vertices. By construction, exactly one vertex of  $D$  is coloured red, and all other vertices of  $D$  are blue.

Let  $S^* = S''$  be the set containing the red vertex of  $D$ . Let  $T^*$  be the singleton set containing the blue neighbour of the vertex in  $S^*$ . Let  $T''$  be the set of blue vertices, so  $T^* \subseteq T''$ . We exhaustively apply rules R1 and R2 on the starting pair  $(S'', T'')$ . By Lemma 13



■ **Figure 5** The situation in Claim 17.3 where two connected components  $Z_1, Z_2$  of  $G[Z]$ , each with at least two vertices, are both coloured red. This will always yield an induced path on at least six vertices, even if  $w_1$  and  $w_2$  are not adjacent, as at most one of  $z'_1, z'_2$  is adjacent to  $w_3$ .

we either find in polynomial time that  $G$  has no valid red-blue  $(S^*, T^*, S'', T'')$ -colouring, and we discard the branch, or we obtain an intermediate tuple  $(S', T', X', Y')$  of  $G$ . Suppose the latter case holds. Let  $Z' = V \setminus (X' \cup Y')$  be the set of uncoloured vertices.

$\triangleright$  **Claim 17.1.** Every vertex  $z \in Z'$  has a neighbour in  $Y' \setminus T'$  that belongs to  $F$ .

**Proof.** As  $F$  is dominating,  $z$  has a neighbour in  $F$ . Since  $D \supseteq V(F)$  contains exactly one red vertex  $x$ , which has a blue neighbour in  $D$ , all neighbours of  $x$  in  $G - D$  are coloured red, that is, belong to  $X$ . As  $z \in G - D$  belongs to  $Z'$ , this means that  $x$  and  $z$  are non-adjacent. So, the neighbour of  $z$  in  $F$  must belong to  $Y' \setminus T'$  (as else we could have applied R2).  $\triangleleft$

$\triangleright$  **Claim 17.2.** The intermediate tuple  $(S', T', X', Y')$  is monochromatic.

**Proof.** Suppose for a contradiction that there is an edge  $uv \in E(G[Z'])$  such that  $u$  is blue and  $v$  is red. Then  $v$  has two blue neighbours by Claim 17.1, a contradiction.  $\triangleleft$

Since Claim 17.2 holds, we may now exhaustively apply R1–R3 to the intermediate tuple  $(S', T', X', Y')$ . By Lemma 15 we either find in polynomial time that  $G$  has no valid red-blue  $(S', T', X', Y')$ -colouring, and thus no valid red-blue  $(S^*, T^*, S', T')$ -colouring, and we discard the branch, or we obtain a final tuple  $(S, T, X, Y)$  of  $G$ . Again, we let  $Z = V \setminus (X \cup Y)$ . By the same lemma and Claim 17.1, the following holds for every (uncoloured) vertex  $w \in Z$ :

- $w$  has at most one neighbour in  $X \setminus S$ ,
- $w$  has exactly one neighbour in  $Y \setminus T$ , which belongs to  $F$ , and
- if  $w'$  is in the same connected component of  $G[Z]$  as  $w$ , then  $w$  and  $w'$  do not share a neighbour in  $G - Z$ .

$\triangleright$  **Claim 17.3.** In any valid red-blue  $(S, T, X, Y)$ -colouring at most one red component may have more than one vertex.

**Proof.** For a contradiction, assume that  $Z_1$  and  $Z_2$  are connected components of size at least 2 that are both coloured red. For  $i = 1, 2$ , let  $z_i$  and  $z'_i$  be two adjacent vertices in  $Z_i$ , and let  $w_i$  be the blue neighbour of  $z_i$  in  $F$  (which exists due to Claim 17.1). Note that  $w_1$  is not adjacent to any vertex of  $\{z'_1, z_2, z'_2\}$ , and  $w_2$  is not adjacent to any vertex of  $\{z_1, z'_1, z'_2\}$ . Moreover,  $w_1$  and  $w_2$  are distinct vertices, and do not have any other neighbours in  $Z_1 \cup Z_2$ . If  $w_1$  and  $w_2$  are adjacent, then  $z'_1 z_1 w_1 w_2 z_2 z'_2$  is an induced  $P_6$ . As  $G$  is  $P_6$ -free, this is not possible. Hence,  $w_1$  and  $w_2$  are not adjacent.

We now use the fact that  $w_1$  and  $w_2$  both belong to  $F$  and that  $F$  is complete bipartite. As  $w_1 w_2 \notin E$ , the latter means there is a vertex  $w_3 \in V(F)$  adjacent to both  $w_1$  and  $w_2$ , so  $w_3$  is blue as well. As  $z'_1$  and  $z'_2$  are both coloured red, at most one of  $z'_1, z'_2$  can be adjacent

to  $w_3$ . Hence, we may assume without loss of generality that  $w_3$  is not adjacent to  $z'_1$ . As  $z_1$  and  $z_2$  have  $w_1$  and  $w_2$ , respectively, as their matching partner,  $w_3$  is adjacent neither to  $z_1$  nor to  $z_2$ . Now,  $z'_1 z_1 w_1 w_3 w_2 z_2$  is an induced  $P_6$ , a contradiction. See also Figure 5.  $\triangleleft$

We then exhaustively apply rules R1-R3 again. This takes polynomial time. In essence, we merely pre-coloured some more vertices red. So, in the end we either find a new tuple of  $G$  with the same properties as those listed in Lemma 15, or we find that  $G$  has not such a tuple, in which case we discard the branch. Suppose we have not discarded the branch. Now the set of uncoloured vertices form an independent set. Hence, we can apply Lemma 16 to find in polynomial time a red-blue colouring of  $G$  that is a maximum red-blue  $(S^*, T^*, S'', T'')$ -colouring due to Lemmas 13-(ii) and 15-(ii).

If somewhere in the above process we discarded a branch, that is, if  $G$  has no valid red-blue  $(S^*, T^*, S'', T'')$ -colouring, we consider the next one. Else remember the value of the maximum red-blue  $(S^*, T^*, S'', T'')$ -colouring that we found. Afterwards, we pick one with the largest value to obtain a maximum valid red-blue colouring of  $G$ .

The correctness of our algorithm follows from its description. The running time is polynomial: each of the in total  $O(n^3)$  branches takes polynomial time to process.  $\blacktriangleleft$

The proof of our next result combines Lemma 16 with arguments from the proof that MATCHING CUT is polynomial-time solvable for  $(H + P_3)$ -free graphs if it is so for  $H$ -free graphs [30]. We omit the proof.

► **Theorem 18.** *Let  $H$  be a graph. If MAXIMUM MATCHING CUT is polynomial-time solvable for  $H$ -free graphs, then it is so for  $(H + P_2)$ -free graphs.*

We now show our third polynomial-time result; we will again apply Lemma 16.

► **Theorem 19.** *MAXIMUM MATCHING CUT is solvable in polynomial time for graphs with diameter at most 2.*

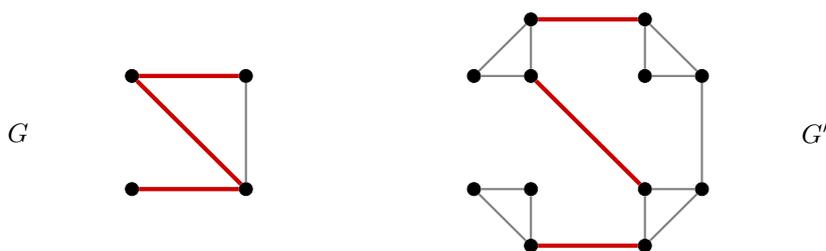
**Proof sketch.** Let  $G = (V, E)$  be a graph of diameter at most 2. If  $G$  has diameter 1, then the problem is trivial to solve. Assume that  $G$  has diameter 2. By Observation 10 it suffices to find a maximum valid red-blue colouring of  $G$ . By definition, such a colouring has at least one bichromatic edge (has value at least 1). We branch over all  $O(n^2)$  options of choosing the bichromatic edge.

Consider a branch, where  $e = uv$  is the bichromatic edge, say  $u$  is blue and  $v$  is red. All other neighbours of  $u$  must be coloured blue. Let  $D = \{u\} \cup N(u)$ . Then  $D$  dominates  $G$ , as  $G$  has diameter 2. Set  $S^* = \{u\}$ ,  $T^* = \{v\}$ ,  $S'' = \{u\}$  and  $T'' = N(u)$ . This gives us a starting pair  $(S'', T'')$  with core  $(S^*, T^*)$ . We exhaustively apply R1 and R2 on  $(S'', T'')$ . By Lemma 13 we either find in polynomial time that  $G$  has no valid red-blue  $(S^*, T^*, S'', T'')$ -colouring, and we discard the branch, or we obtain an intermediate tuple  $(S', T', X', Y')$ . Say the latter holds. Let  $Z' = V \setminus (X' \cup Y')$  be the set of uncoloured vertices. We show the following claim (proof omitted).

▷ **Claim 19.1.** The intermediate tuple  $(S', T', X', Y')$  is monochromatic.

By Claim 19.1, we may exhaustively apply R1–R3 to the intermediate tuple  $(S', T', X', Y')$ . By Lemma 15 we either find in polynomial time that  $G$  has no valid red-blue  $(S', T', X', Y')$ -colouring, and thus no valid red-blue  $(S^*, T^*, S', T')$ -colouring, and we discard the branch, or we obtain a final tuple  $(S, T, X, Y)$  of  $G$ . We let  $Z = V \setminus (X \cup Y)$ . We show the following claims (proofs omitted).

▷ **Claim 19.2.** Every vertex  $w \in Z$  has exactly one neighbour in  $X \setminus S$ .



■ **Figure 6** A graph  $G$  (left) where the thick red edges form a maximum edge cut, and the graph  $G'$  (right) from the proof of Theorem 20, where the thick red edges form a maximum matching cut.

▷ **Claim 19.3.** If  $G[Z]$  contains two connected components  $F_1$  and  $F_2$  of size at least 2, then  $G[Z] = F_1 + F_2$ .

We use Claim 19.2 to prove Claim 19.3, from which it follows that  $G[Z]$  has at most two components with more than one vertex, which are both monochromatic in every valid red-blue  $(S, T, X, Y)$ -colouring of  $G$  (if such a colouring exists) due to Claim 19.1. Hence, we can branch over all possible colourings of these connected components (there are at most four branches).

For each branch, we propagate the obtained partial red-blue colouring by exhaustively applying rules R1–R3. This takes polynomial time. In essence, we merely pre-coloured some more vertices red or blue. So, in the end we either find a new tuple of  $G$  with the same properties as those listed in Lemma 15, or we find that  $G$  has not such a tuple, in which case we discard the branch. Suppose not. Now the set of uncoloured vertices form an independent set. Hence, we can apply Lemma 16 to find in polynomial time a red-blue colouring of  $G$  that is a maximum red-blue  $(S^*, T^*, S'', T'')$ -colouring due to Lemmas 13-(ii) and 15-(ii).

If we discarded a branch, that is, if  $G$  has no valid red-blue  $(S^*, T^*, S'', T'')$ -colouring, we consider the next one. If we did not discard the branch, then we remember the value of the maximum red-blue  $(S^*, T^*, S'', T'')$ -colouring that we found. Afterwards, we pick one with the largest value to obtain a maximum valid red-blue colouring of  $G$ .

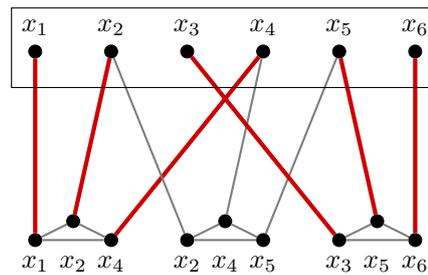
The correctness of our branching algorithm follows from its description. The running time is polynomial: each of the in total  $O(n^2)$  branches takes polynomial time to process. ◀

## 4 Hardness Results

We sketch the two hardness proofs. For the first one we reduce from MAXIMUM CUT, which is NP-complete even for subcubic graphs [37]: does a subcubic graph  $G$  have an edge cut of size at least  $k$  for some integer  $k$ ?

► **Theorem 20.** MAXIMUM MATCHING CUT is NP-hard for subcubic line graphs of triangle-free graphs.

**Proof sketch.** Let  $(G, k)$  be an instance of MAXIMUM CUT, where  $G$  is subcubic. Build a graph  $G'$  as follows (see also Figure 6). Replace every vertex  $v \in V(G)$  by a triangle  $C_v$ . For every edge  $uv \in E(G)$ , add an edge between a vertex in  $C_v$  and a vertex in  $C_u$ , such that every vertex in  $C_v$  has at most one neighbour outside  $C_v$ . This is possible, as  $G$  is subcubic. The graph  $G'$  is also subcubic, as every vertex in  $G'$  has two neighbours inside a triangle and at most one neighbour outside. Moreover,  $G'$  is  $(K_{1,3}, \text{diamond})$ -free, so the line graph of a triangle-free graph. We can show that  $G$  has an edge cut of size at least  $k$  if and only if  $G'$  has a maximum matching cut of size at least  $k$ . ◀



■ **Figure 7** The graph  $G$  for  $X = \{x_1, \dots, x_6\}$  and  $\mathcal{S} = \{\{x_1, x_2, x_4\}, \{x_2, x_4, x_5\}, \{x_3, x_5, x_6\}\}$ . The vertices in the rectangle form a clique. The set  $\mathcal{S}' = \{\{x_1, x_2, x_4\}, \{x_3, x_5, x_6\}\}$  is an exact 3-cover of  $X$ . The thick red edges in the graph show the corresponding maximum matching cut.

An *exact 3-cover* of a set  $X$  is a collection  $\mathcal{C}$  of 3-element subsets of  $X$ , such that every  $x \in X$  is in exactly one 3-element subset of  $\mathcal{C}$ . We now reduce from EXACT 3-COVER, which is to decide if a collection  $\mathcal{S}$  of 3-element subsets of a set  $X$  with  $q$  elements has an *exact 3-cover* of  $X$  (which will be of size  $q$ ). This problem is NP-complete (see [23]).

► **Theorem 21.** MAXIMUM MATCHING CUT is NP-hard for  $2P_3$ -free quadrangulated graphs of radius at most 2 and diameter at most 3.

**Proof sketch.** Let  $(X, \mathcal{S})$  be an instance of EXACT 3-COVER where  $X = \{x_1, \dots, x_{3q}\}$  and  $\mathcal{S} = \{S_1, \dots, S_k\}$ , such that each  $S_i$  contains exactly three elements of  $X$ . From  $(X, \mathcal{S})$  we construct a graph  $G$  as follows (see also Figure 7). We first define a clique  $K_X = \{x_1, \dots, x_{3q}\}$ . For each  $S \in \mathcal{S}$ , we do as follows. Let  $S = \{x_h, x_i, x_j\}$ . We add a triangle  $K_S$  on vertices  $x_h^S, x_i^S$  and  $x_j^S$ . We add an edge between a vertex  $x_i \in K_X$  and a vertex  $u \notin K_X$  if and only if  $u = x_i^S$  for some  $S \in \mathcal{S}$ . This completes the construction of  $G$ . Note that  $G$  is  $2P_3$ -free, quadrangulated as well as that the radius of  $G$  is at most 2 and the diameter of  $G$  is at most 3. We can show that  $\mathcal{S}$  contains an exact 3-cover of  $X$  if and only if  $G$  has a matching cut of size  $3q$ . ◀

## 5 The Proofs of Theorems 5–7

We first note that Theorem 5 immediately follows from Theorems 19 and 21.

The first part of Theorem 6 follows from the fact that a graph of radius 1 has a dominating vertex, and thus it has a matching cut if and only if it has a vertex of degree 1, which can be checked in polynomial time. The second part of Theorem 6 follows from Theorem 21.

To prove Theorem 7, let  $H$  be a graph. If  $H$  contains a cycle, then MATCHING CUT, and thus MAXIMUM MATCHING CUT, is NP-hard due to Theorem 2. Now suppose  $H$  has no cycle, so  $H$  is a forest. If  $H$  contains a vertex of degree at least 3, then the class of  $H$ -free graphs contains the class of  $K_{1,3}$ -free graphs. The latter class contains the class of line graphs, and thus we apply Theorem 20. Now suppose  $H$  is a forest of maximum degree at most 2, that is,  $H$  is a linear forest. If  $H \subseteq_i sP_2 + P_6$  for some  $s \geq 0$ , then we apply Theorem 17. Else  $H$  has an induced  $2P_3$ . We apply Theorem 21. This completes the proof.

## 6 Conclusions

We considered the optimization version MAXIMUM MATCHING CUT of the classical MATCHING CUT problem after first observing that the PERFECT MATCHING CUT problem is a special case of the former problem. We generalized known algorithms for graphs of diameter at most 2

and  $P_6$ -free graphs from MATCHING CUT and PERFECT MATCHING CUT to MAXIMUM MATCHING CUT. We also showed that the latter problem is computationally harder (assuming  $P \neq NP$ ) than MATCHING CUT and PERFECT MATCHING CUT for various graph classes. Our results led to three new dichotomy results, including a complete computational complexity classification of MAXIMUM MATCHING CUT for  $H$ -free graphs. The latter classification is still unsettled for the other two problems, as can be observed from Theorems 2 and 4. Below we discuss some other open problems.

We first recall that the complexity of PERFECT MATCHING CUT has not been fully classified for graphs of diameter at most  $d$ . What is the complexity of PERFECT MATCHING CUT in the remaining open case where  $d = 3$ ? We showed that MAXIMUM MATCHING CUT is NP-hard for  $2P_3$ -free quadrangulated graphs of diameter 3 and radius 2, whereas MATCHING CUT is polynomial-time solvable for quadrangulated graphs [32]. We recall an interesting open problem of Le and Telle [29] who asked, after proving polynomial-time solvability for chordal graphs: what is the complexity of PERFECT MATCHING CUT for quadrangulated graphs, or more general,  $k$ -chordal graphs for  $k \geq 4$ ? Here, a graph is  $k$ -chordal for some  $k \geq 3$  if it is  $(C_{k+1}, C_{k+2}, \dots)$ -free, so 3-chordal graphs are the chordal graphs.

---

## References

- 1 Júlio Araújo, Nathann Cohen, Frédéric Giroire, and Frédéric Havet. Good edge-labelling of graphs. *Discrete Applied Mathematics*, 160:2502–2513, 2012.
- 2 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. Vertex partitioning problems on graphs with bounded tree width. *Discrete Applied Mathematics*, 319:254–270, 2022.
- 3 N. R. Aravind and Roopam Saxena. An FPT algorithm for Matching Cut and  $d$ -Cut. *Proc. IWCCA 2021, LNCS*, 12757:531–543, 2021.
- 4 Edouard Bonnet, Dibyayan Chakraborty, and Julien Duron. Cutting barnette graphs perfectly is hard. *Proc. WG 2023, LNCS*, to appear.
- 5 Paul S. Bonsma. The complexity of the Matching-Cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62:109–126, 2009 (conference version: WG 2003).
- 6 Mieczyslaw Borowiecki and Katarzyna Jesse-Józefczyk. Matching cutsets in graphs of diameter 2. *Theoretical Computer Science*, 407:574–582, 2008.
- 7 Valentin Bouquet and Christophe Picouleau. The complexity of the Perfect Matching-Cut problem. *CoRR*, abs/2011.03318, 2020.
- 8 Chi-Yeh Chen, Sun-Yuan Hsieh, Hoàng-Oanh Le, Van Bang Le, and Sheng-Lung Peng. Matching Cut in graphs with large minimum degree. *Algorithmica*, 83:1238–1255, 2021.
- 9 Maria Chudnovsky. The structure of bull-free graphs II and III – A summary. *Journal of Combinatorial Theory, Series B*, 102:252–282, 2012.
- 10 Maria Chudnovsky and Paul Seymour. The structure of claw-free graphs. *Surveys in Combinatorics, London Mathematical Society Lecture Note Series*, 327:153–171, 2005.
- 11 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8:51–53, 1984.
- 12 Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Clique-width for hereditary graph classes. *Proc. BCC 2019, London Mathematical Society Lecture Note Series*, 456:1–56, 2019.
- 13 Arthur M. Farley and Andrzej Proskurowski. Networks immune to isolated line failures. *Networks*, 12:393–403, 1982.
- 14 Carl Feghali. A note on Matching-Cut in  $P_t$ -free graphs. *Information Processing Letters*, 179:106294, 2023.
- 15 Carl Feghali, Felicia Lucke, Daniël Paulusma, and Bernard Ries. Matching cuts in graphs of high girth and  $H$ -free graphs. *CoRR*, abs/2212.12317, 2022.

- 16 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84:331–363, 2017.
- 17 Petr A. Golovach, Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Refined notions of parameterized enumeration kernels with applications to matching cut enumeration. *Journal of Computer and System Sciences*, 123:76–102, 2022.
- 18 Petr A. Golovach, Daniël Paulusma, and Jian Song. Computing vertex-surjective homomorphisms to partially reflexive trees. *Theoretical Computer Science*, 457:86–100, 2012.
- 19 Guilherme Gomes and Ignasi Sau. Finding cuts of bounded degree: complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83:1677–1706, 2021.
- 20 Ronald L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York Academy of Sciences*, 175:170–186, 1970.
- 21 Pinar Heggeres and Jan Arne Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5:128–142, 1998.
- 22 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. *ACM Transactions on Algorithms*, 15:25:1–25:90, 2019.
- 23 Richard M. Karp. Reducibility among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.
- 24 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching Cut: Kernelization, single-exponential time FPT, and exact exponential algorithms. *Discrete Applied Mathematics*, 283:44–58, 2020.
- 25 Dieter Kratsch and Van Bang Le. Algorithms solving the Matching Cut problem. *Theoretical Computer Science*, 609:328–335, 2016.
- 26 Hoang-Oanh Le and Van Bang Le. A complexity dichotomy for Matching Cut in (bipartite) graphs of fixed diameter. *Theoretical Computer Science*, 770:69–78, 2019.
- 27 Hoàng-Oanh Le and Van Bang Le. Complexity results for matching cut problems in graphs without long induced paths. *Proc. WG 2023, LNCS*, to appear.
- 28 Van Bang Le and Bert Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301:463–475, 2003.
- 29 Van Bang Le and Jan Arne Telle. The Perfect Matching Cut problem revisited. *Theoretical Computer Science*, 931:117–130, 2022.
- 30 Felicia Lucke, Daniël Paulusma, and Bernard Ries. On the complexity of Matching Cut for graphs of bounded radius and  $H$ -free graphs. *Theoretical Computer Science*, 936, 2022.
- 31 Felicia Lucke, Daniël Paulusma, and Bernard Ries. Finding matching cuts in  $H$ -free graphs. *Algorithmica*, to appear.
- 32 Augustine M. Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13:527–536, 1989.
- 33 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the Matching-Cut problem. *Proc. WG 2001, LNCS*, 2204:284–295, 2001.
- 34 Ján Plesník. Constrained weighted matchings and edge coverings in graphs. *Discrete Applied Mathematics*, 92:229–241, 1999.
- 35 Bert Randerath and Ingo Schiermeyer. Vertex colouring and forbidden subgraphs – A survey. *Graphs and Combinatorics*, 20:1–40, 2004.
- 36 Pim van’t Hof and Daniël Paulusma. A new characterization of  $P_6$ -free graphs. *Discrete Applied Mathematics*, 158:731–740, 2010.
- 37 Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. *Proc. STOC 1978*, pages 253–264, 1978.



# A Weyl Criterion for Finite-State Dimension and Applications

Jack H. Lutz  

Department of Computer Science, Iowa State University, Ames, IA, USA

Satyadev Nandakumar  

Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur, U.P., India

Subin Pulari  

Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur, U.P., India

---

## Abstract

Finite-state dimension, introduced early in this century as a finite-state version of classical Hausdorff dimension, is a quantitative measure of the lower asymptotic *density of information* in an infinite sequence over a finite alphabet, as perceived by finite automata. Finite-state dimension is a robust concept that now has equivalent formulations in terms of finite-state gambling, lossless finite-state data compression, finite-state prediction, entropy rates, and automatic Kolmogorov complexity. The 1972 Schnorr-Stimm dichotomy theorem gave the first automata-theoretic characterization of normal sequences, which had been studied in analytic number theory since Borel defined them in 1909. This theorem implies, in present-day terminology, that a sequence (or a real number having this sequence as its base- $b$  expansion) is normal if and only if it has finite-state dimension 1. One of the most powerful classical tools for investigating normal numbers is the 1916 Weyl's criterion, which characterizes normality in terms of exponential sums. Such sums are well studied objects with many connections to other aspects of analytic number theory, and this has made use of Weyl's criterion especially fruitful. This raises the question whether Weyl's criterion can be generalized from finite-state dimension 1 to arbitrary finite-state dimensions, thereby making it a quantitative tool for studying data compression, prediction, etc. i.e., *Can we characterize all compression ratios using exponential sums?*

This paper does exactly this. We extend Weyl's criterion from a characterization of sequences with finite-state dimension 1 to a criterion that characterizes every finite-state dimension. This turns out *not* to be a routine generalization of the original Weyl criterion. Even though exponential sums may diverge for non-normal numbers, finite-state dimension can be characterized in terms of the *dimensions* of the *subsequence limits* of the exponential sums. In case the exponential sums are convergent, they converge to the Fourier coefficients of a probability measure whose *dimension* is precisely the finite-state dimension of the sequence.

This new and surprising connection helps us bring Fourier analytic techniques to bear in proofs in finite-state dimension, yielding a new perspective. We demonstrate the utility of our criterion by substantially improving known results about preservation of finite-state dimension under arithmetic. We strictly generalize the results by Aistleitner and Doty, Lutz and Nandakumar for finite-state dimensions under arithmetic operations. We use the method of exponential sums and our Weyl criterion to obtain the following new result: *If  $y$  is a number having finite-state strong dimension 0, then  $\dim_{FS}(x + qy) = \dim_{FS}(x)$  and  $\text{Dim}_{FS}(x + qy) = \text{Dim}_{FS}(x)$  for any  $x \in \mathbb{R}$  and  $q \in \mathbb{Q}$ .* This generalization uses recent estimates obtained in the work of Hochman [17] regarding the entropy of convolutions of probability measures.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Information theory

**Keywords and phrases** Finite-state dimension, Finite-state compression, Weyl's criterion, Exponential sums, Normal numbers

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.65

**Related Version** *arXiv Version*: <https://arxiv.org/abs/2111.04030>



© Jack H. Lutz, Satyadev Nandakumar, and Subin Pulari;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 65; pp. 65:1–65:16

Leibniz International Proceedings in Informatics



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

**Funding** *Jack H. Lutz*: This author’s research was supported in part by National Science Foundation grant 1900716.

**Acknowledgements** The authors would like to thank Michael Hochman for technical clarifications regarding his paper [17]. We also thank the anonymous reviewers for their valuable comments and suggestions.

## 1 Introduction

Finite-state compressibility [34], or equivalently, finite-state dimension [10, 2, 5] is a quantification of the information rate in data as measured by finite-state automata. This formulation, initially motivated by practical constraints, has proved to be rich and mathematically robust, having several equivalent characterizations. In particular, the finite state-dimension of a sequence is equal to the compression ratio of the sequence using information lossless finite-state compressors ([10, 2]). Finite-state dimension has unexpected connections to areas such as number theory, information theory, and convex analysis [20, 12]. Schnorr and Stimm [28] establish a particularly significant connection by showing that a number is Borel normal in base  $b$  (see for example, [24, 6, 8]) if and only if its base  $b$  expansion has finite-state compressibility equal to 1, *i.e.*, is incompressible (see also: [3, 5, 13, 15]). Equivalently, a number  $x \in [0, 1)$  is normal if and only if  $\dim_{FS}(x)$ , the finite-state dimension of  $x$  is equal to 1. A celebrated characterization of Borel normality in terms of exponential sums, provided by Weyl’s criterion [32], has proved to be remarkably effective in the study of normality. Weyl’s criterion on uniformly distributed sequences modulo 1 yields a characterization that a real number  $r$  is normal to base  $b$  if and only if for every integer  $k$ ,

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k (b^j r)} = 0. \quad (1)$$

This tool was used by Wall [31] in his pioneering thesis to show that normality is preserved under certain operations like selection of subsequences along arithmetic progressions, and multiplication with non-zero rationals. Weyl’s criterion facilitates the application of tools from Fourier analysis in the study of Borel normality. Weyl’s criterion is used in several important constructions of normal numbers including those given by Cassels [7], Erdős and Davenport [11] etc. The criterion was also instrumental in obtaining the construction of absolutely normal numbers given by Schmidt in [27].

The finite-state compression ratio/dimension of an arbitrary sequence is a quantity in  $[0,1]$ . The classical Weyl’s criterion provides a characterization of numbers having finite-state dimension equal to 1 in terms of exponential sums. This leads us to the natural question - *Can we characterize arbitrary compression ratios using exponential sums?* This question turns out to be highly non-trivial. It is not easy to generalize Weyl’s criterion to study arbitrary finite-state compression ratios/dimension. The major conceptual hurdle arises from the fact that for non-normal numbers, the Weyl sum averages in (1) need not converge. The Weyl averages need not converge even when the finite-state dimension and the strong dimension of a sequence are equal.

We demonstrate this by explicitly constructing such a sequence in Lemma 19. Using a new construction method involving the controlled concatenation of two special sequences, we demonstrate the existence of a sequence  $x \in \Sigma^\infty$  with non-convergent Weyl averages, while having finite-state dimension and strong dimension both equal to  $\frac{1}{2}$ . The proof that this constructed sequence satisfies the required properties uses new techniques, which might be

of independent interest. Due to the existence of such sequences, it is unclear how to *extract* the finite-state dimension of a sequence from non-convergent Weyl averages. Indeed, it was unclear whether any generalization of the Weyl's criterion to arbitrary finite-state dimensions even exists.

Our paper rescues this approach and gives such a characterization of arbitrary finite-state compressibility/dimension by introducing one important viewpoint, that turns out to be the major theoretical insight. Even when the exponential sums diverge, the theory of *weak convergence* of probability measures ([4]) enables us to consider the collection of all probability measures having Fourier coefficients equal to the subsequence limits of the Weyl averages. The *dimensions* of the measures in the set of subsequence weak limit measures gives a generalization of Weyl's criterion. For any  $x$ , let  $\dim_{FS}(x)$  and  $\text{Dim}_{FS}(x)$  denote the finite-state dimension and finite-state strong dimension [2] of  $x$  respectively. We now informally state our Weyl's criterion for finite-state dimension.

► **Theorem** (Informal statement of Theorem 22). *Let  $x \in [0, 1)$ . If for any subsequence  $\langle n_m \rangle_{m=0}^\infty$  of natural numbers, there exist complex numbers  $c_k$  such that for every  $k \in \mathbb{Z}$ ,  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k (b^j x)} = c_k$ , then, there exists a probability measure  $\mu$  on  $[0, 1)$  such that for every  $k$ ,  $c_k = \int e^{2\pi i k y} d\mu$ . Let  $\mathcal{W}_x$  be the collection of all such probability measures  $\mu$  on  $[0, 1)$  that can be obtained as the subsequence limits of Weyl averages. Then,  $\dim_{FS}(x) = \inf_{\mu \in \mathcal{W}_x} H^-(\mu)$  and  $\text{Dim}_{FS}(x) = \sup_{\mu \in \mathcal{W}_x} H^+(\mu)$ .*

The *correct* notion of *dimensions* of the subsequence weak limit measures in  $\mathcal{W}_x$  which yields the finite-state dimensions of  $x$  turns out to be  $H^-$  and  $H^+$ , the *lower* and *upper average entropies* of  $\mu$  as defined in [2]<sup>1</sup>. Therefore, this new characterization enables us to *extract* the finite-state compressibility/dimension by studying the behavior of the Weyl sum averages, thereby extending Weyl's criterion for normality to arbitrary finite-state dimensions.

An interesting special case of our criterion is when the exponential averages of a sequence are convergent. In this case, the averages  $\langle c_k \rangle_{k \in \mathbb{Z}}$  are precisely the *Fourier coefficients* of a *unique* limiting measure, whose *dimension* is precisely the finite-state dimension of the sequence. This relates two different notions of dimension to each other. We give the informal statement of our criterion for this special case.

► **Theorem** (Informal statement of Theorem 23). *Let  $x \in [0, 1)$ . If there exist complex numbers  $c_k$  for  $k \in \mathbb{Z}$  such that  $\frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k (b^j x)} \rightarrow c_k$  as  $n \rightarrow \infty$ , then, there exists a unique measure  $\mu$  on  $[0, 1)$  such that for every  $k$ ,  $c_k = \int e^{2\pi i k y} d\mu$ . Furthermore,  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = H^-(\mu) = H^+(\mu)$ .*

Our results also show that in case there is a unique weak limit measure, the exponential sums (1) converge for every  $k \in \mathbb{Z}$ . These give the first known relations between Fourier coefficients and finite-state compressibility/dimension. The proof of Weyl's criterion for finite-state dimension is not a routine generalization of the available proofs of Weyl's criterion for normality (see [32, 14], [30]) and requires several facts from the theory of weak convergence of probability measures and new relationships involving the exponential sums, the *dimensions* of weak limit measures and the finite-state dimension of the given sequence. We overcome certain additional technical difficulties in working with two different topologies - the topology on the torus  $\mathbb{T}$  where Fourier coefficients uniquely determine a measure, and another, Cantor space, which is required for studying combinatorial properties of sequences, like normality.

<sup>1</sup> These are analogues of the well-known *Rényi upper and lower dimensions* of measures as defined in [25]. See the remark following Definition 5.

## 1.1 Applications of our criterion

We illustrate how this framework can be applied in sections 6. These results justify that this framework pioneers a new, powerful, approach to data compression. It is not very surprising that when the Weyl averages converge, our criterion has applications. Importantly, even in situations where the Weyl averages do not converge, it is possible to derive non-trivial consequences. We apply our techniques to substantially improve known results about the preservation of finite-state dimension under arithmetic and combinatorial operations.

Doty, Lutz and Nandakumar [12] show that if  $x$  is any real and  $q$  is any non-zero rational, then the finite-state dimensions and strong dimensions of  $x$ ,  $qx$  and  $x + q$  are equal. When  $x$  is normal, a generalization is obtained by Aistleitner [1], which can be described as follows. Let  $y$  be any real such that the asymptotic density of zeroes in its expansion is one. Then, for any rational  $q$ , we have  $x + qy$  is normal. We generalize these results by allowing both the following conditions simultaneously,

1.  $x$  is allowed to be any real, obtaining a result for all finite-state dimensions rather than only for normals as in Aistleitner [1] and
2.  $y$  is allowed to be any real with finite-state strong dimension 0 which satisfies a natural independence condition. This generalizes both the restrictions in Doty, Lutz and Nandakumar [12] and Aistleitner [1],

and show that for any rational  $q \in \mathbb{Q}$ ,  $\dim_{FS}(x + qy) = \dim_{FS}(x)$  and  $\text{Dim}_{FS}(x + qy) = \text{Dim}_{FS}(x)$  if  $x$  and  $y$  are *independent* (see Definition 30) and  $\text{Dim}_{FS}(y) = 0$ .

Using our Weyl criterion along with the results in Hochman [17], we obtain the following. Let  $x$  and  $y$  be real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent. Then for any  $d, e \in \mathbb{Z}$ ,  $\dim_{FS}(dx + ey) \geq \max\{\dim_{FS}(dx), \dim_{FS}(ey)\}$  and  $\dim_{FS}(dx + ey) \leq \dim_{FS}(dx) + \text{Dim}_{FS}(ey)$ . Similarly,  $\text{Dim}_{FS}(dx + ey) \geq \max\{\text{Dim}_{FS}(dx), \text{Dim}_{FS}(ey)\}$  and  $\text{Dim}_{FS}(dx + ey) \leq \text{Dim}_{FS}(dx) + \text{Dim}_{FS}(ey)$ . Our main results are consequences of these inequalities.

There are several known techniques for explicit constructions of normal numbers (see [20, 6]), but constructions of those with finite-state dimension  $s \in [0, 1)$  follow two techniques: first, to start with a normal sequence, and to *dilute* it with an appropriate fraction of simple patterns, as we did in Section 4, and second, to start with a coin with bias  $p$  such that  $-p \log_2 p - (1 - p) \log_2 (1 - p) = s$ , and consider any typical sequence drawn from this distribution (see also [23]). We note that our Weyl criterion along with techniques from Mance and Madritsch [22] yields new methods for the explicit construction of numbers having a specified finite-state dimension.

Lossless data compression is practically significant, and theoretically sophisticated. We show how one of the major tools of modern mathematics, Fourier analysis, can be brought to bear to study compressibility of individual data sequences. We hope that our criterion will facilitate the application of more powerful Fourier analytic tools in future works involving finite-state compression/dimension.

After the preliminary sections, section 3 gives Weyl's criterion on Cantor space using weak convergence of measures. Next, we show the necessity and the sufficiency of passing to *subsequences* of sequences of measures in order to generalize Weyl's criterion for finite-state dimension. In section 6 we show the applications of our Weyl criterion to yield new, general results regarding the preservation of finite-state dimension under arithmetic and combinatorial operations.

## 2 Preliminaries

For any natural number  $b > 1$ ,  $\Sigma_b$  denotes the alphabet  $\{0, 1, 2, \dots, b-1\}$ . Throughout this paper, we work with base 2, but our results generalize to all bases. We use  $\Sigma$  to denote the binary alphabet  $\Sigma_2$ . We denote the set of finite binary strings by  $\Sigma^*$  and the set of infinite sequences by  $\Sigma^\infty$ . For any  $w \in \Sigma^*$ , let  $C_w$  be the set of infinite sequences with  $w$  as a prefix, called a *cylinder*. For any sequence  $x = x_0x_1x_2\dots$  in  $\Sigma^\infty$ , we denote the substring  $x_ix_{i+1}\dots x_j$  of  $x$ , by  $x_i^j$ . The Borel  $\sigma$ -algebra generated by the set of all cylinder sets is denoted by  $\mathcal{B}(\Sigma^\infty)$ . Let  $\mathbb{T}$  denote the one-dimensional torus or unit circle. i.e.,  $\mathbb{T}$  is the unit interval  $[0, 1)$  with the metric  $d(r, s) = \min\{|r - s|, 1 - |r - s|\}$ .  $\mathbb{T}$  is a compact metric space. The Borel  $\sigma$ -algebra generated by all open sets in  $\mathbb{T}$  is denoted by  $\mathcal{B}(\mathbb{T})$ . For any base  $b$ , let  $v_b$  be the *evaluation map* which maps any  $x \in \Sigma^\infty$  to its *value* in  $\mathbb{T}$  which is  $\sum_{i=0}^\infty \frac{x_i}{b^{i+1}} \bmod 1$ . We use the simplified notation  $v$  to denote the base 2 evaluation map  $v_2$ . Let  $T$  be the left shift transformation  $T(x_0x_1x_2\dots) = x_1x_2x_3\dots$  on  $\Sigma^\infty$ . For any base  $b$  and  $w \in \Sigma_b^*$ , let  $I_w^b$  denote the interval  $[v_b(w0^\infty), v_b(w0^\infty) + b^{-|w|})$  in  $\mathbb{T}$ . We use the simplified notation  $I_w$  to refer to  $I_w^2$ . Let  $\mathbb{D}$  be the set of all dyadic rationals in  $\mathbb{T}$ . It is easy to see that  $v : \Sigma^\infty \rightarrow \mathbb{T}$  has a well-defined inverse, denoted  $v^{-1}$ , over  $\mathbb{T} \setminus \mathbb{D}$ . For any measure  $\mu$  on  $\mathbb{T}$  (or  $\Sigma^\infty$ ), we refer to the collection of complex numbers  $\int e^{2\pi iky} d\mu$  where  $k$  ranges over  $\mathbb{Z}$  as the *Fourier coefficients of measure*  $\mu$ . For measures over  $\Sigma^\infty$ , the function  $e^{2\pi iky}$  inside the integral is replaced with  $e^{2\pi ikv(y)}$ . For every measure  $\mu$  on  $\mathbb{T}$ , we define the corresponding lifted measure on  $\Sigma^\infty$  as follows.

► **Definition 1** (Lift  $\hat{\mu}$  of a measure  $\mu$  on  $\mathbb{T}$ ). *If  $\mu$  is a measure on  $\mathbb{T}$ , then we define the lift  $\hat{\mu}$  of  $\mu$  to be the unique measure on  $\Sigma^\infty$  satisfying  $\hat{\mu}(C_w) = \mu(I_w)$  for every string  $w \in \Sigma^*$ .*<sup>2</sup>

► **Definition 2.** *Let  $x \in \Sigma^*$  have length  $n$ . We define the sliding count probability of  $w \in \Sigma^*$  in  $x$  denoted  $P(x, w)$ , and the disjoint block probability of  $w$  in  $x$ , denoted  $P^d(x, w)$ , as follows.*

$$P(x, w) = \frac{|\{i \in [0, n - |w|] : x_i^{i+|w|-1} = w\}|}{n - |w| + 1} \quad \text{and} \quad P^d(x, w) = \frac{|\{i \in [0, \frac{n}{|w|}] : x_{|w|i}^{|w|(i+1)-1} = w\}|}{n/|w|}$$

Now, we define normal sequences in  $\Sigma^\infty$  and normal numbers on  $\mathbb{T}$ .

► **Definition 3.** *A sequence  $x \in \Sigma^\infty$  is normal if for every  $w \in \Sigma^*$ ,  $\lim_{n \rightarrow \infty} P(x_0^{n-1}, w) = 2^{-|w|}$ .  $r \in \mathbb{T}$  is normal if and only if  $r \notin \mathbb{D}$  and  $v^{-1}(r)$  is a normal sequence in  $\Sigma^\infty$ .*

Equivalently, we can formulate normality using disjoint probabilities [20]. The following is the block entropy characterization of finite-state dimension from [5], which we use instead of the original formulation using  $s$ -gales (see [10],[2],[21]).

► **Definition 4** ([10, 5]). *For a given block length  $l$ , we define the sliding block entropy over  $x_0^{n-1}$  as  $H_l(x_0^{n-1}) = -\frac{1}{l} \sum_{w \in \Sigma^l} P(x_0^{n-1}, w) \log(P(x_0^{n-1}, w))$ . The finite-state dimension of  $x \in \Sigma^\infty$ , denoted  $\dim_{FS}(x)$ , and finite-state strong dimension of  $x$ , denoted  $\text{Dim}_{FS}(x)$ , are defined as follows.  $\dim_{FS}(x) = \inf_l \liminf_{n \rightarrow \infty} H_l(x_0^{n-1})$  and  $\text{Dim}_{FS}(x) = \inf_l \limsup_{n \rightarrow \infty} H_l(x_0^{n-1})$ .*<sup>3</sup>

<sup>2</sup> The uniqueness of  $\hat{\mu}$  follows from routine measure theoretic arguments

<sup>3</sup> The fact that  $\dim_{FS}(x)$  and  $\text{Dim}_{FS}(x)$  are equivalent to the lower and upper finite-state compressibilities of  $x$  using lossless finite-state compressors, follows immediately from the results in [34] and [10].

Disjoint block entropy  $H_l^d$  is defined similarly by replacing  $P$  with  $P^d$ . Bourke, Hitchcock and Vinodchandran [5], based on the work of Ziv and Lempel [34], demonstrated the entropy characterization of finite-state dimension using  $H_l^d$  instead of  $H_l$ . Kozachinskiy and Shen ([19]) proved that the finite-state dimension of a sequence can be equivalently defined using sliding block entropies (as in Definition 4) instead of disjoint block entropies. It is clear from the definition that, for any  $x \in \Sigma^\infty$ ,  $\dim_{FS}(x) \leq \text{Dim}_{FS}(x)$ . Any  $x$  with  $\dim_{FS}(x) = \text{Dim}_{FS}(x)$  is called a *regular sequence*. Upper and lower average entropies were defined in [2] for measures constructed out of infinite bias sequences. We extend these notions to the set of all measures on  $\Sigma^\infty$  below.

► **Definition 5.** For any probability measure  $\mu$  on  $\Sigma^\infty$ , let  $\mathbf{H}_n(\mu) = -\sum_{w \in \Sigma^n} \mu(C_w) \log(\mu(C_w))$ . The upper average entropy of  $\mu$ , denoted  $H^+(\mu)$ , and its lower average entropy, denoted  $H^-(\mu)$ , are respectively the limit superior and the limit inferior as  $n$  tends to  $\infty$  of  $H_n(\mu)/n$ .

Upper and lower average entropies are the Cantor space analogues of Rényi upper and lower dimensions of measures on  $[0,1]$  which were originally defined for measures on the real line in [25]. For any  $x \in \mathbb{T}$  (or  $x \in \Sigma^\infty$ ), let  $\delta_x$  denote the Dirac measure at  $x$ . i.e,  $\delta_x(A) = 1$  if  $x \in A$  and 0 otherwise for every  $A \in \mathcal{B}(\mathbb{T})$  (or  $A \in \mathcal{B}(\Sigma^\infty)$ ). Given a sequence  $\langle x_n \rangle_{n=0}^\infty$  of numbers in  $\mathbb{T}$  (or  $\Sigma^\infty$ ), we investigate the behavior of exponential averages  $\frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k x_j}$  by studying the weak convergence of sequences of averages of Dirac measures.

► **Definition 6.** Given a sequence  $\langle x_n \rangle_{n=0}^\infty$  in  $\mathbb{T}$  (or elements in  $\Sigma^\infty$ ), we say that  $\langle \nu_n \rangle_{n=1}^\infty$  is the sequence of averages of Dirac measures over  $\mathbb{T}$  (or over  $\Sigma^\infty$ ) constructed out of the sequence  $\langle x_n \rangle_{n=0}^\infty$  if,  $\nu_n = n^{-1} \sum_{i=0}^{n-1} \delta_{x_i}$  for each  $n \in \mathbb{N}$ .

### 3 Weyl's criterion and weak convergence

Schnorr and Stimm [28] (see also [3, 5]) showed a central connection between normal numbers and finite-state compressibility, or equivalently, finite-state dimension: a sequence  $x \in \Sigma^\infty$  is normal if and only if its finite-state dimension is 1. Any  $x \in \Sigma^\infty$  has finite-state dimension (equivalently, finite-state compressibility) between 0 and 1. In this sense, finite-state dimension is a generalization of the notion of normality. Another celebrated characterization of normality, in terms of exponential sums, was provided by Weyl in 1916. This characterization has resisted attempts at generalization. In the present section, we show that the theory of weak convergence of measures yields a generalization of Weyl's characterization for arbitrary dimensions. We demonstrate the utility of this new characterization to finite-state compressibility/finite-state dimension, in subsequent sections. Weyl criterion for normal numbers on  $\mathbb{T}$  is the following.

► **Theorem 7** (Weyl's criterion [32]). A number  $r \in \mathbb{T}$  is normal if and only if for every  $k \in \mathbb{Z}$ ,  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k (2^j r)} = 0$ .

The insight in this theorem is the connection between a number  $x$  being normal, and the concept of the collection of its shifts being uniformly distributed in the unit interval. It is the latter concept which leads to the cancellation of the exponential sums of all orders. The following is a formulation of this criterion on Cantor space, which we require in our work.

► **Theorem 8** (Weyl's criterion on  $\Sigma^\infty$ ). A sequence  $x \in \Sigma^\infty$  is a normal sequence if and only if for every  $k \in \mathbb{Z}$ ,  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k (v(T^j x))} = 0$ .

The key to generalizing Weyl’s criterion to sequences with finite-state dimension less than 1 is to characterize convergence of subsequences of exponential sums using weak convergence of probability measures on  $\Sigma^\infty$  (see Billingsley [4]). Over  $\mathbb{T}$ , this equivalent characterization is well-known (see Section 4.4 from [14]). Obtaining the same equivalence over  $\Sigma^\infty$  involves some technical hurdles due to the fact that continuous functions over  $\Sigma^\infty$  need not have a uniform approximation using trigonometric polynomials. In order to overcome these, we need to carefully study the relationship between the convergence of Weyl averages and weak convergence over  $\Sigma^\infty$ . We develop these relationships in the following lemmas. At the end of this section we characterize Theorem 8 in terms of weak convergence of a sequence of measures over  $\Sigma^\infty$ .

► **Definition 9.** A sequence  $\langle \nu_n \rangle_{n \in \mathbb{N}}$  of probability measures on a metric space  $(X, d)$  converges weakly to a probability  $\mu$  on  $(X, d)$ , denoted  $\nu_n \Rightarrow \mu$ , if for every bounded continuous function  $f : X \rightarrow \mathbb{C}$ , we have  $\lim_{n \rightarrow \infty} \int f d\nu_n = \int f d\mu$ .

If a sequence of measures  $\langle \nu_n \rangle_{n \in \mathbb{N}}$  on a metric space  $(X, d)$  has a weak limit measure, then the weak limit must be unique (see Theorem 1.2 from [4]). Since  $\mathbb{T}$  and  $\Sigma^{\infty 4}$  are compact metric spaces, using Prokhorov’s Theorem (see Theorem 5.1 from [4]) we get that any sequence of measures  $\langle \nu_n \rangle_{n \in \mathbb{N}}$  on  $\mathbb{T}$  (or  $\Sigma^\infty$ ), has a measure  $\mu$  on  $\mathbb{T}$  (or  $\Sigma^\infty$ ) and a subsequence  $\langle \nu_{n_m} \rangle_{m \in \mathbb{N}}$  such that  $\nu_{n_m} \Rightarrow \mu$ . We first establish a relationship between weak convergence of measures on  $\mathbb{T}$  and the convergence of measures of dyadic intervals in  $\mathbb{T}$ . Since the set of all finite unions of dyadic intervals in  $\mathbb{T}$  is closed under finite intersections, we obtain the following lemma using Theorem 2.2 from [4].

► **Lemma 10.** If for every dyadic interval  $I$  in  $\mathbb{T}$ ,  $\lim_{n \rightarrow \infty} \nu_n(I) = \mu(I)$ , then  $\nu_n \Rightarrow \mu$ .

The Portmanteau theorem (Theorem 2.1 from [4]) gives the following partial converse.

► **Lemma 11.** Let  $\nu_n \Rightarrow \mu$ . Then  $\lim_{n \rightarrow \infty} \nu_n(I) = \mu(I)$  for dyadic interval  $I = [d_1, d_2)$  if  $\mu(\{d_1\}) = \mu(\{d_2\}) = 0$ .

We characterize convergence of exponential sums in terms of weak convergence of probability measures, first on  $\mathbb{T}$  and then on the Cantor space  $\Sigma^\infty$ . Unlike Theorem 7, the result on  $\Sigma^\infty$  does not follow immediately from that on  $\mathbb{T}$ . On  $\mathbb{T}$ , the following theorem holds due to Prokhorov theorem, the fact that continuous functions on  $\mathbb{T}$  can be approximated uniformly using trigonometric polynomials, and that Fourier coefficients of measures over  $\mathbb{T}$  are unique due to Bochner’s theorem (see Theorem 4.19 from [16]).

► **Theorem 12.** Let  $r \in \mathbb{T}$  and let  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures constructed out of  $\langle 2^n r \bmod 1 \rangle_{n=0}^\infty$ . Let  $\langle n_m \rangle_{m \in \mathbb{N}}$  be any subsequence of natural numbers. Then for every  $k \in \mathbb{Z}$ , there is a  $c_k \in \mathbb{C}$  such that  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k (2^j r)} = c_k$  if and only if there is a unique measure  $\mu$  such that  $\nu_{n_m} \Rightarrow \mu$ . Furthermore, if any of the above conditions are true, then  $c_k = \int e^{2\pi i k y} d\mu$  for every  $k \in \mathbb{Z}$  and  $\mu$  is the unique measure on  $\mathbb{T}$  having Fourier coefficients  $\langle c_k \rangle_{k \in \mathbb{Z}}$ .

We require an analogue of this theorem for Cantor space. But the proof above cannot be adapted because on Cantor space, there are continuous functions which cannot be approximated uniformly using trigonometric polynomials. For example, consider  $\chi_{C_0}$ . Observe that  $\chi_{C_0}(0^\infty) = 1 \neq 0 = \chi_{C_0}(1^\infty)$ . But since  $v(0^\infty) = v(1^\infty)$ , every trigonometric polynomial

<sup>4</sup> The metric on  $\Sigma^\infty$  is  $d(x, y) = 2^{-\min\{i | x_i \neq y_i\}}$ .

has the same value on  $0^\infty$  and  $1^\infty$ . However, we recover the analogue by handling dyadic rational sequences and other sequences in separate cases. Since the set of all finite unions of cylinder sets in  $\Sigma^\infty$  is closed under finite intersections and since the characteristic functions of cylinder sets are continuous on the Cantor space, we get the following analogue of Lemma 10 and 11 using Theorem 2.2 from [4].

► **Lemma 13.** *For a sequence of measures  $\langle \nu_n \rangle_{n \in \mathbb{N}}$  on  $\Sigma^\infty$ ,  $\nu_n \Rightarrow \mu$  iff  $\lim_{n \rightarrow \infty} \nu_n(C_w) = \mu(C_w)$  for every  $w \in \Sigma^*$ .*

In the following theorems we relate the convergence of measures of cylinder sets to the convergence of Weyl averages on the Cantor space using Theorem 12 and Lemma 13. We state these theorems for convergence along any subsequence, since we require these more general results for studying the subsequence limits of Weyl averages.

► **Theorem 14.** *Let  $x \in \Sigma^\infty$  and  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures on  $\Sigma^\infty$  constructed out of  $\langle T^n x \rangle_{n=0}^\infty$ . Let  $\langle n_m \rangle_{m \in \mathbb{N}}$  be any subsequence of natural numbers. If  $\lim_{m \rightarrow \infty} \nu_{n_m}(C_w) = \mu(C_w)$  for every  $w \in \Sigma^*$ , then for every  $k \in \mathbb{Z}$  we have  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k v(T^j x)} = \int e^{2\pi i k v(y)} d\mu$ .*

Observe that  $\frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k v(T^j x)} = \int e^{2\pi i k v(y)} d\nu_{n_m}$ . Hence, the above claim follows from Lemma 13 and the definition of weak convergence since for every  $k \in \mathbb{Z}$ ,  $e^{2\pi i k v(y)}$  is a continuous function on  $\Sigma^\infty$ <sup>5</sup>. While Fourier coefficients uniquely determine measures over  $\mathbb{T}$ , Bochner's Theorem does not hold over  $\Sigma^\infty$ . For example let  $\mu_1 = \delta_{0^\infty}$  and let  $\mu_2 = \delta_{1^\infty}$ . Then  $\mu_1 \neq \mu_2$ , but it is easy to verify that for any  $k \in \mathbb{Z}$ ,  $\int e^{2\pi i k v(y)} d\mu_1 = e^{2\pi i k v(0^\infty)} = 1 = \int e^{2\pi i k v(y)} d\mu_2$ . The following lemma leads to a converse of Theorem 14.

► **Lemma 15.** *Let  $x \in \Sigma^\infty$  such that  $v(x) \notin \mathbb{D}$  and let  $\langle \nu'_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures on  $\mathbb{T}$  constructed out of the sequence  $\langle 2^n v(x) \bmod 1 \rangle_{n=0}^\infty$ . Let  $d$  be any non-zero dyadic rational. If  $\nu'_{n_m} \Rightarrow \mu'$  for some subsequence of natural numbers  $\langle n_m \rangle_{m \in \mathbb{N}}$ , then  $\mu'(\{d\}) = 0$ .*

Using the above results we obtain the following partial converse of Theorem 14.

► **Theorem 16.** *Let  $x \in \Sigma^\infty$  and let  $\langle n_m \rangle_{m \in \mathbb{N}}$  be any subsequence of natural numbers. Let  $\langle c_k \rangle_{k \in \mathbb{Z}}$  be complex numbers such that  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k v(T^j x)} = c_k$  for every  $k \in \mathbb{Z}$ . Then there exists a unique measure  $\mu$  on  $\mathbb{T}$  having Fourier coefficients  $\langle c_k \rangle_{k \in \mathbb{Z}}$  and  $\lim_{m \rightarrow \infty} \nu_{n_m}(C_w) = \hat{\mu}(C_w)$  for every  $w \in \Sigma^*$  such that  $w \neq 1^{|w|}$  and  $w \neq 0^{|w|}$ .*

For any  $x \in \Sigma^\infty$ , let  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures on  $\Sigma^\infty$  constructed out of the sequence  $\langle T^n x \rangle_{n=0}^\infty$ . Now, for any  $A \in \mathcal{B}(\Sigma^\infty)$ ,  $\nu_n(A)$  is the proportion of elements in the finite sequence  $x, Tx, T^2x, \dots, T^{n-1}x$  which falls inside the set  $A$ . From this remark, and the definitions of  $\nu_n$  and the sliding count probability  $P$ , the following lemma follows easily.

► **Lemma 17.** *Let  $w$  be any finite string in  $\Sigma^*$  and let  $l = |w|$ . Let  $x$  be any element in  $\Sigma^\infty$ . If  $\langle \nu_n \rangle_{n=1}^\infty$  is the sequence of averages of Dirac measures over  $\Sigma^\infty$  constructed out of the sequence  $\langle T^n x \rangle_{n=0}^\infty$ . Then for any  $n$ ,  $\nu_n(C_w) = P(x_0^{n+l-2}, w)$ .*

We now give a new characterization of Weyl's criterion on Cantor Space (Theorem 8) in terms of weak convergence of measures.

<sup>5</sup> This follows easily by observing that the valuation map  $v : \Sigma^\infty \rightarrow \mathbb{T}$  is a continuous function on  $\Sigma^\infty$ .

► **Theorem 18** (Weyl's criterion on  $\Sigma^\infty$  and weak convergence). *Let  $x \in \Sigma^\infty$ , and  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures constructed out of  $\langle T^n x \rangle_{n=0}^\infty$ , and  $\mu$  be the uniform measure on  $\Sigma^\infty$ . Then the following are equivalent.*

1.  $x$  is normal.
2. For every  $w \in \Sigma^*$ , the sliding block frequency  $P(x_0^{n-1}, w) \rightarrow 2^{-|w|}$  as  $n \rightarrow \infty$ .
3. For every  $k \in \mathbb{Z}$ ,  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k v(T^j x)} = 0$ .
4.  $\nu_n \Rightarrow \mu$ .

#### 4 Divergence of exponential sums for non-normal numbers

Weyl's criterion says that when  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = 1$  the averages of the exponential sums for every  $k$  converges to 0. However for  $x$  with  $\dim_{FS}(x) < 1$ , the situation is different. It is easy to construct a sequence  $a$  with  $\dim_{FS}(a) < 1$  and a  $k \in \mathbb{Z}$  such that the sequence of Weyl averages with parameter  $k$  do not converge. It is natural to ask if the condition  $\dim_{FS}(x) = \text{Dim}_{FS}(x)$  is sufficient to guarantee convergence of the exponential sum averages. But we construct an  $x$  with  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = \frac{1}{2}$  such that for some  $k$ , the sequence  $\langle \sum_{j=0}^{n-1} e^{2\pi i k v(T^j x)} \rangle_{n=1}^\infty$  diverges. Entropy rates converging to a limit does not imply that the empirical probability measures converge to a limiting distribution, and it is the latter notion which is necessary for exponential sums to converge.

► **Lemma 19.** *There exists  $x \in \Sigma^\infty$  with  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = \frac{1}{2}$  such that for some  $k \in \mathbb{Z}$ , the sequence  $\langle \sum_{j=0}^{n-1} e^{2\pi i k v(T^j x)} \rangle_{n=1}^\infty$  is not convergent.*

Generalizing the construction of diluted sequences in [10], we define an  $x$  with  $v(x) \in \mathbb{T} \setminus \mathbb{D}$  and  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = 1/2$ , but where for some  $k \in \mathbb{Z}$ , the sequences of Weyl sum averages diverge. The idea of dilution is as follows. Let  $y \in \Sigma^\infty$  be normal. Define  $a \in \Sigma^\infty$  by  $a_{2n} = 0$ ,  $a_{2n+1} = y_n$ ,  $n \in \mathbb{N}$ . Then  $\dim_{FS}(a) = \text{Dim}_{FS}(a) = 1/2$ . Note that  $b \in \Sigma^\infty$  defined by  $b_{4n} = b_{4n+3} = 0$ , and  $b_{4n+1} = y_{2n}$ ,  $b_{4n+2} = y_{2n+1}$ ,  $n \in \mathbb{N}$  is also a regular sequence with  $\dim_{FS}(b) = \text{Dim}_{FS}(b) = 1/2$ . But, the sliding block frequency of 01 in  $a$  is  $1/4$ , whereas it is  $3/16$  in  $b$ . We leverage the existence of such distinct sequences with equal dimension. The disjoint blocks of  $x$  alternate between the above two patterns in a controlled manner to satisfy the following conditions.

1.  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = 1/2$
2. There is an increasing sequence of indices  $\langle n_i \rangle_{i=1}^\infty$  such that  $\lim_{i \rightarrow \infty} P(x_0^{n_i-1}, 01) = 1/4$ .
3. There is an increasing sequence of indices  $\langle n_j \rangle_{j=1}^\infty$  such that  $\lim_{j \rightarrow \infty} P(x_0^{n_j-1}, 01) = 3/16$ .

Let  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures constructed out of  $\langle T^n x \rangle_{n=0}^\infty$ , and  $\langle \nu'_n \rangle_{n=1}^\infty$ , those from  $\langle 2^n v(x) \bmod 1 \rangle_{n=0}^\infty$ . Assume that  $\langle n^{-1} \sum_{j=0}^{n-1} e^{2\pi i k v(T^j x)} \rangle_{n=1}^\infty$  converge for every  $k \in \mathbb{Z}$ . Using the same steps in the proof of Theorem 16, we get that  $\nu'_n \Rightarrow \mu'$  where  $\mu'$  is the unique measure on  $\mathbb{T}$  having Fourier coefficients equal to the limits of the Weyl averages. Since  $v(x) \in \mathbb{T} \setminus \mathbb{D}$ , Theorem 16 implies that  $\nu(C_{01})$  is convergent. Using Lemma 17, we infer that  $\lim_{n \rightarrow \infty} P(x_0^{n-1}, 01)$  exists. But, we know from conditions 2 and 3 that  $P(x_0^{n-1}, 01)$  is not convergent. Hence, we arrive at a contradiction. Therefore, for some  $k \in \mathbb{Z}$ , the Weyl averages  $\langle n^{-1} \sum_{j=0}^{n-1} e^{2\pi i k v(T^j x)} \rangle_{n=1}^\infty$  diverge. The above construction is easily adapted to show that for any rational number  $p/q \in (0, 1)$ , there exists  $x \in \Sigma^\infty$  with  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = p/q$  such that some Weyl average of  $x$  diverges.

## 5 Weyl's criterion for finite-state dimension

We saw in Lemma 19 that Weyl averages may diverge for  $x$  having finite-state dimension less than 1, even if  $x$  is regular. Hence, it is necessary for us to deal with divergent Weyl averages and obtain their relationship with the finite-state dimension of  $x$ . We know from Theorem 18 that Weyl's criterion for normality (Theorem 8) is equivalently expressed in terms of weak convergence of a sequence of measures over  $\Sigma^\infty$ . In section 5.1, we generalize the weak convergence formulation to handle arbitrary finite state dimension. Applying this, in section 5.2, we generalize the exponential sum formulation.

### 5.1 Weak convergence and finite-state dimension

We know from Theorem 18 that  $x \in \Sigma^\infty$  is normal (equivalently,  $\dim_{FS}(x) = 1$ ) if and only if  $\nu_n \rightarrow \mu$ , where  $\mu$  is the uniform distribution over  $\Sigma^\infty$ . In this subsection we give a generalization of this formulation of Weyl's criterion which applies for  $x$  having any finite-state dimension. Lemma 19 and Theorem 14 together imply that  $\nu_n$ 's need not be weakly convergent even if  $x$  is guaranteed to be regular. However, studying the subsequence limits of  $\langle \nu_n \rangle_{n=1}^\infty$  gives us the following generalization of Weyl's criterion for arbitrary  $x \in \Sigma^\infty$ .

► **Theorem 20.** *Let  $x \in \Sigma^\infty$ . Let  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures on  $\Sigma^\infty$  constructed out of the sequence  $\langle T^n x \rangle_{n=0}^\infty$ . Let  $\mathcal{W}_x$  be the collection of all subsequence weak limits of  $\langle \nu_n \rangle_{n=1}^\infty$ . i.e.,  $\mathcal{W}_x = \{\mu \mid \exists \langle n_m \rangle_{m=0}^\infty \text{ such that } \nu_{n_m} \Rightarrow \mu\}$ . Then,  $\dim_{FS}(x) = \inf_{\mu \in \mathcal{W}_x} H^-(\mu)$  and  $\text{Dim}_{FS}(x) = \sup_{\mu \in \mathcal{W}_x} H^+(\mu)$ .*

The following is an equivalent version of Theorem 20 which we require in section 5.2. From the definition of lower average entropy, Theorem 20 shows that,  $\dim_{FS}(x) = \inf_{\mu \in \mathcal{W}_x} \liminf_{l \rightarrow \infty} \mathbf{H}_l(\mu)/l$ . This  $\liminf$  can be replaced by an infimum.

► **Lemma 21.**  $\dim_{FS}(x) = \inf_{\mu \in \mathcal{W}_x} \inf_l \mathbf{H}_l(\mu)/l$

### 5.2 Weyl averages and finite-state dimension

We now obtain the main result of the paper by relating subsequence limits of Weyl averages and finite-state dimension. In case the Weyl averages converge, we show that the sequence is regular. In particular, when the Weyl averages converge to 0, then the regular sequence is normal. We know from Lemma 19 that there exist regular sequences with non-convergent Weyl averages. In the absence of limits, we investigate the subsequence limits of Weyl averages in order to obtain a relationship with the finite-state dimension. If for some  $x \in \Sigma^\infty$ , there exist a sequence of natural numbers  $\langle n_m \rangle_{m \in \mathbb{N}}$  and constants  $\langle c_k \rangle_{k \in \mathbb{Z}}$  such that  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k v(T^j x)} = c_k$ . Then, using Theorem 16, we get that there exists a measure  $\mu$  on  $\mathbb{T}$  such that  $c_k = \int e^{2\pi i k y} d\mu$  and  $\lim_{m \rightarrow \infty} \nu_{n_m}(C_w) = \hat{\mu}(C_w)$  for every  $w \neq 0^{|w|}$  and  $w \neq 1^{|w|}$ . But,  $\nu_{n_m}(C_{0^l})$  and  $\nu_{n_m}(C_{1^l})$  need not converge. Simple examples of such strings can be obtained by concatenating increasingly large runs of 0's and 1's in an alternating stage wise manner. However, the probabilities of the strings  $0^l$  and  $1^l$  have negligible effect on the finite-state dimension as  $l$  gets large. Using Theorem 20 we obtain the following.

► **Theorem 22** (Weyl's criterion for finite-state dimension). *Let  $x \in \Sigma^\infty$ . If for any  $\langle n_m \rangle_{m=0}^\infty$  there exist constants  $c_k$  for  $k \in \mathbb{Z}$  such that  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k v(T^j x)} = c_k$ , for every  $k \in \mathbb{Z}$ , then there exists a measure  $\mu$  on  $\mathbb{T}$  such that for every  $k$ ,  $c_k = \int e^{2\pi i k y} d\mu$ . Let  $\widehat{\mathcal{W}}_x$  be the collection of the lifted measures  $\hat{\mu}$  on  $\Sigma^\infty$  for all  $\mu$  on  $\mathbb{T}$  that can be obtained as subsequence limits of Weyl averages. Then,  $\dim_{FS}(x) = \inf\{H^-(\hat{\mu}) \mid \hat{\mu} \in \widehat{\mathcal{W}}_x\}$  and  $\text{Dim}_{FS}(x) = \sup\{H^+(\hat{\mu}) \mid \hat{\mu} \in \widehat{\mathcal{W}}_x\}$*

**Proof of Theorem 22.** If  $v(x)$  is a dyadic rational in  $\mathbb{T}$ , then it can be easily verified that the Weyl averages are convergent to 1. The unique measure having all Fourier coefficients equal to 1 over  $\mathbb{T}$  is  $\delta_0$ . Since  $\tilde{\delta}_0 = \delta_{0^\infty}$ , it can be easily verified that  $\dim_{FS}(x) = H^-(\delta_{0^\infty}) = H^+(\delta_{0^\infty}) = \text{Dim}_{FS}(x) = 0$ . Hence, we consider the case when  $v(x)$  is not a dyadic rational. We first define analogues of finite-state dimension by avoiding the strings  $0^l$  and  $1^l$  for all  $l$  in calculating the sliding entropies. We define  $\tilde{H}_l(x_0^{n-1})$  to be the normalized sliding entropy over  $x_0^{n-1}$  as in the definition of  $H_l(x_0^{n-1})$ , except that the summation is taken over  $\Sigma^l \setminus \{0^l, 1^l\}$  instead of  $\Sigma^l$ . Using this notion, we define  $\widetilde{\dim}_{FS}(x) = \liminf_{l \rightarrow \infty} \liminf_{n \rightarrow \infty} \tilde{H}_l(x_0^{n-1})$  and  $\widetilde{\text{Dim}}_{FS}(x) = \liminf_{l \rightarrow \infty} \limsup_{n \rightarrow \infty} \tilde{H}_l(x_0^{n-1})$ . Since,  $\tilde{H}_l(x_0^{n-1}) \leq H_l(x_0^{n-1}) \leq \tilde{H}_l(x_0^{n-1}) + 2/l$ , it can be shown using routine arguments that  $\dim_{FS}(x) = \widetilde{\dim}_{FS}(x)$  and  $\text{Dim}_{FS}(x) = \widetilde{\text{Dim}}_{FS}(x)$ . Similarly we define  $\tilde{H}^+$  and  $\tilde{H}^-$  by reducing the range of the sum in the definition of  $H_l$  to  $\Sigma^l \setminus \{0^l, 1^l\}$  instead of  $\Sigma^l$ . Using a similar argument as in the case of sliding entropy, it can be shown that  $\tilde{H}^+$  and  $\tilde{H}^-$  are the same as  $H^+$  and  $H^-$  for any measure on  $\Sigma^\infty$ . Let  $\langle \nu_n \rangle_{n=1}^\infty$  be the sequence of averages of Dirac measures on  $\Sigma^\infty$  constructed out of the sequence  $\langle T^n x \rangle_{n=0}^\infty$ . Let  $\mathcal{W}_x$  be the set of all weak limits of  $\nu_n$  as constructed in Theorem 20. Since  $v(x)$  is not a dyadic rational, using Prokhorov's theorem for weak convergence of  $\mathbb{T}$  and weak convergence over  $\Sigma^\infty$ , it can be shown that,  $\inf_{\mu \in \mathcal{W}_x} H^-(\mu) = \inf_{\hat{\mu} \in \widehat{\mathcal{W}}_x} H^-(\hat{\mu})$  and  $\sup_{\mu \in \mathcal{W}_x} H^+(\mu) = \sup_{\hat{\mu} \in \widehat{\mathcal{W}}_x} H^+(\hat{\mu})$ . The claim now follows from Theorem 20. ◀

Hence, the finite-state dimension and finite-state strong dimension are related to the lower and upper average entropies of the subsequence limits of the Weyl averages. Using the above result, we get the following theorem in the case when the Weyl averages are convergent.

► **Theorem 23** (Weyl's criterion for convergent Weyl averages). *Let  $x \in \Sigma^\infty$ . If there exist  $c_k \in \mathbb{C}$  for  $k \in \mathbb{Z}$  such that  $\frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k(v(T^j x))} \rightarrow c_k$  as  $n \rightarrow \infty$ , then, there exists a unique measure  $\mu$  on  $\mathbb{T}$  such that for every  $k$ ,  $c_k = \int e^{2\pi i k y} d\mu$ . Furthermore,  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = H^-(\hat{\mu}) = H^+(\hat{\mu})$ .*

As a special case, we derive Weyl's criterion for normality, i.e, for sequences  $x$  such that  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = 1$  as a special case of Theorem 20 and Theorem 23.

► **Theorem 24.** *Let  $x \in \Sigma^\infty$ . Then  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k(v(T^j x))} = 0$  for every  $k \in \mathbb{Z}$  if and only  $\dim_{FS}(x) = \text{Dim}_{FS}(x) = 1$ .*

The conclusion of Theorem 23 says that  $\dim_{FS}(x) = \text{Dim}_{FS}(x)$ . i.e,  $x$  is a regular sequence. Hence, Lemma 19 and Theorem 23 together yield the following.

► **Corollary 25.** *If for each  $k \in \mathbb{Z}$ ,  $\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k(v(T^j x))} = c_k$  for a sequence of complex numbers  $\langle c_k \rangle_{k \in \mathbb{Z}}$ . Then,  $x$  is a regular sequence. But there exist regular sequences having non-convergent Weyl averages.*

## 6 Preservation of finite-state dimension under real arithmetic

In this section, we demonstrate the utility of our framework by proving the most general results yet regarding the preservation of finite-state dimension under arithmetic operations like addition with reals satisfying a natural independence condition, and multiplication with non-zero rationals. These results strictly generalize all known results regarding the preservation of finite-state dimension including those of Doty, Lutz and Nandakumar [12] and Aistleitner [1]. Our Weyl criterion plays a pivotal role in these extensions. We combine our Weyl criterion along with recent estimates by Hochman [17] for the entropy of convolution of probability measures. It is easier to analyze addition and multiplication as operations

over  $\mathbb{T}$ . Hence we first obtain an equivalent Weyl's criterion for finite-state dimension in terms of measures over  $\mathbb{T}$ . We now define the analogues of upper and lower average entropies for measures over  $\mathbb{T}$ . This turns out to be the notion of Rényi dimension as defined by Alfréd Rényi in [25]. Recall that for any  $m$  and  $w \in \Sigma_m^n$ ,  $I_w^m$  denotes the interval  $[v_m(w0^\infty), v_m(w0^\infty) + m^{-|w|}]$  in  $\mathbb{T}$ .

► **Definition 26** (Rényi Dimension). *For any probability measure  $\mu$  on  $\mathbb{T}$  and partition factor  $m$ , let  $\mathbf{H}_n^m(\mu) = -\sum_{w \in \Sigma_m^n} \mu(I_w^m) \log(\mu(I_w^m))$ . The Rényi upper and lower dimensions (see [25] and [33]) are defined as follows,  $\overline{\dim}_R^m(\mu) = \limsup_{n \rightarrow \infty} \frac{\mathbf{H}_n^m(\mu)}{n \log m}$  and  $\underline{\dim}_R^m(\mu) = \liminf_{n \rightarrow \infty} \frac{\mathbf{H}_n^m(\mu)}{n \log m}$ . If  $\overline{\dim}_R^m(\mu) = \underline{\dim}_R^m(\mu)$  then the Rényi dimension of  $\mu$  is  $\dim_R^m(\mu) = \overline{\dim}_R^m(\mu) = \underline{\dim}_R^m(\mu)$ .*

From the above definition, it seems as if the notion of Rényi dimension is dependent on the choice of the partition factor  $m$ . However, Rényi upper and lower dimensions are quantities that are independent of the partition factor<sup>6</sup>. Hence, we suppress the partition factor  $m$  in the notations  $\overline{\dim}_R^m(\mu)$ ,  $\underline{\dim}_R^m(\mu)$  and  $\dim_R^m(\mu)$  and use  $\overline{\dim}_R(\mu)$ ,  $\underline{\dim}_R(\mu)$  and  $\dim_R(\mu)$  to refer to the corresponding quantities for a measure  $\mu$  on  $\mathbb{T}$ . Now, we state an equivalent Weyl's criterion for finite-state dimension for  $r \in \mathbb{T}$  in terms of weak limit measures over  $\mathbb{T}$  and Rényi dimension of measures over  $\mathbb{T}$ .

► **Theorem 27** (Restatement of Weyl's criterion for finite-state dimension (Theorem 22)). *Let  $r \in \mathbb{T}$ . If for any  $\langle n_m \rangle_{m=0}^\infty$  there exist  $c_k$  for  $k \in \mathbb{Z}$  such that  $\lim_{m \rightarrow \infty} \frac{1}{n_m} \sum_{j=0}^{n_m-1} e^{2\pi i k 2^j r} = c_k$  for every  $k \in \mathbb{Z}$ , then there exists a measure  $\mu$  on  $\mathbb{T}$  such that for every  $k$ ,  $c_k = \int e^{2\pi i k y} d\mu$ . Let  $\mathcal{W}_r$  be the collection of all  $\mu$  on  $\mathbb{T}$  that can be obtained as subsequence limits of Weyl averages. Then,  $\dim_{FS}(r) = \inf\{\underline{\dim}_R(\mu) \mid \mu \in \mathcal{W}_r\}$  and  $\text{Dim}_{FS}(r) = \sup\{\overline{\dim}_R(\mu) \mid \mu \in \mathcal{W}_r\}$ .*

D. D. Wall in his thesis [31] proved that if  $r \in [0, 1]$  and  $q$  is any non-zero rational number, then  $r$  is a normal number if and only if  $qr$  and  $q + r$  are normal numbers. Doty, Lutz and Nandakumar [12] generalized this result to arbitrary finite-state dimensions and proved that the finite-state dimension and finite-state strong dimension of any number are preserved under multiplication and addition with rational numbers.

► **Theorem 28** ([12]). *Let  $r \in \mathbb{T}$  and  $q$  be any non-zero rational number. Then for any base  $b$ ,  $\dim_{FS}^b(r) = \dim_{FS}^b(q + r) = \dim_{FS}^b(qr)$  and  $\text{Dim}_{FS}^b(r) = \text{Dim}_{FS}^b(q + r) = \text{Dim}_{FS}^b(qr)$ .*

In the above  $\dim^b$  and  $\text{Dim}^b$  denotes the finite-state dimension and finite-state strong dimension of the number  $r$  calculated by considering the sequence representing the base- $b$  expansion of  $r$ <sup>8</sup>. In the specific case of normal sequences, Wall's result has been generalized by Aistleitner in the following form. Let  $\mathcal{C}$  be the set of reals  $y = 0.y_0y_1\dots$  such that the ratio  $P(y_0^{n-1}, 0)$  goes to 1 as  $n$  tends to  $\infty$ . Then we have the following.

► **Theorem 29**. *If  $y \in \mathcal{C}$ , then for any normal  $r \in \mathbb{T}$  and  $q \in \mathbb{Q}$ , the number  $r + qy$  is normal.*

We strictly generalize all these above results by formulating a natural independence notion between two reals. We describe the framework below. Given strings  $x$  and  $y$  in  $\Sigma^\infty$  and strings  $u, w \in \Sigma^\ell$  for some  $\ell \geq 1$ , we define the *joint occurrence count* of  $u$  and  $w$

<sup>6</sup> This important fact regarding Rényi dimension is a folklore result.

<sup>7</sup> The  $2^j$  term in limit expression must be replaced with  $b^j$  while investigating the above criterion in any arbitrary base  $b$

<sup>8</sup> For  $r$  having multiple base  $b$  expansions, this does not cause any ambiguity since in this case the finite-state dimensions of  $r$  are 0 with respect to any of the two possible expansions.

in  $x$  and  $y$  up to  $n$  as,  $N_{u,w}(x_0^{n-1}, y_0^{n-1}) = |\{i \in [0, n - \ell] : x_i^{i+\ell-1} = u \text{ and } y_i^{i+\ell-1} = w\}|$ . And, then the *joint occurrence probability* of  $u$  and  $w$  in  $x$  and  $y$  up to  $n$  is defined as  $P_{u,w}(x_0^{n-1}, y_0^{n-1}) = \frac{N_{u,w}(x_0^{n-1}, y_0^{n-1})}{n-\ell+1}$ .

Informally, we define two infinite strings  $x$  and  $y$  to be independent if for infinitely many lengths  $l$ , the occurrence probability distributions of  $l$ -length strings within  $x$  and  $y$  are *independent* in the limit. The straightforward formulation of independence between  $x$  and  $y$  is  $\lim_{n \rightarrow \infty} P_{u,w}(x_0^{n-1}, y_0^{n-1}) = \lim_{n \rightarrow \infty} P(x_0^{n-1}, u)P(y_0^{n-1}, w)$ . But these limits need not exist for general  $x$  and  $y$ . Hence, the more admissible and useful definition is the following.

► **Definition 30.** Any two strings  $x$  and  $y$  in  $\Sigma^\infty$  are said to be independent if for infinitely many  $\ell \geq 1$  and for every  $u, w \in \Sigma^\ell$ ,  $\lim_{n \rightarrow \infty} |P_{u,w}(x_0^{n-1}, y_0^{n-1}) - P(x_0^{n-1}, u)P(y_0^{n-1}, w)| = 0$ .

For any measures  $\mu_1$  and  $\mu_2$  on  $\mathbb{T}$ , let  $\mu_1 * \mu_2$  denote the convolution of these two measures (see [26] or [16]). A basic intuition for our approach can be viewed as follows. A standard result in probability theory (see for example, Shiryaev [29], 2nd. edition, Section II.8) is that, if  $X$  and  $Y$  are two independent random variables, then the distribution of  $X + Y$  is the convolution of the distributions of  $X$  and  $Y$ . Moreover, the Fourier coefficients of the convolution is the product of the Fourier coefficients of the individual distributions. Our result may be viewed as an analogous result using sequences. The following theorem gives an important connection between the exponential averages of the sum of independent reals and the exponential averages of the individual reals which is crucial in proving the main results in this section.

► **Theorem 31.** If  $x$  and  $y$  are real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30, then for any integers  $d, e$  and  $q \in \mathbb{Q}$ ,

$$\lim_{n \rightarrow \infty} \left| \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k 2^j (dx+ey)} - \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k 2^j dx} \frac{1}{n} \sum_{j=0}^{n-1} e^{2\pi i k 2^j ey} \right| = 0.$$

The proofs of the following bounds on Rényi dimension of convolutions crucially employ results from Hochman [17] along with the data processing inequality ([9, 18]).

► **Lemma 32.** For any measures  $\mu_1$  and  $\mu_2$  on  $\mathbb{T}$ ,

1.  $\underline{\dim}_R(\mu_1 * \mu_2) \geq \max\{\underline{\dim}_R(\mu_1), \underline{\dim}_R(\mu_2)\}$  and  $\underline{\dim}_R(\mu_1 * \mu_2) \leq \underline{\dim}_R(\mu_1) + \overline{\dim}_R(\mu_2)$
2.  $\overline{\dim}_R(\mu_1 * \mu_2) \geq \max\{\overline{\dim}_R(\mu_1), \overline{\dim}_R(\mu_2)\}$  and  $\overline{\dim}_R(\mu_1 * \mu_2) \leq \overline{\dim}_R(\mu_1) + \underline{\dim}_R(\mu_2)$

The following is our main result.

► **Theorem 33.** Let  $x$  and  $y$  be real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30. Then for any  $d, e \in \mathbb{Z}$ ,

1.  $\dim_{FS}(dx + ey) \geq \max\{\dim_{FS}(dx), \dim_{FS}(ey)\}$  and  $\dim_{FS}(dx + ey) \leq \dim_{FS}(dx) + \text{Dim}_{FS}(ey)$ .
2.  $\text{Dim}_{FS}(dx + ey) \geq \max\{\text{Dim}_{FS}(dx), \text{Dim}_{FS}(ey)\}$  and  $\text{Dim}_{FS}(dx + ey) \leq \text{Dim}_{FS}(dx) + \dim_{FS}(ey)$ .

The following technical lemmas that are consequences of Theorem 31 are required for proving Theorem 33.

► **Lemma 34.** Let  $x$  and  $y$  be real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30 and let  $d, e \in \mathbb{Z}$ . Then, for any  $\mu \in \mathcal{W}_{dx+ey}$  there exist  $\mu_1 \in \mathcal{W}_{dx}$  and  $\mu_2 \in \mathcal{W}_{ey}$  such that  $\mu = \mu_1 * \mu_2$ .

► **Lemma 35.** *If  $x$  and  $y$  are real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30. Let  $d, e \in \mathbb{Z}$  and  $q \in \mathbb{Q}$ . Then, for any  $\mu_1 \in \mathcal{W}_{dx}$  there exist  $\mu \in \mathcal{W}_{dx+ey}$  and  $\mu_2 \in \mathcal{W}_{ey}$  such that  $\mu = \mu_1 * \mu_2$ .*

Now we prove Theorem 33.

**Proof of Theorem 33.** Consider any  $\mu \in \mathcal{W}_{dx+ey}$ . Using Lemma 34 we get that there exists  $\mu_1 \in \mathcal{W}_{dx}$  and  $\mu_2 \in \mathcal{W}_{ey}$  such that  $\mu = \mu_1 * \mu_2$ . Now, it follows from Lemma 32 that  $\underline{\dim}_R(\mu) = \underline{\dim}_R(\mu_1 * \mu_2) \geq \underline{\dim}_R(\mu_1)$ . On applying Theorem 27 for  $dx \in \mathbb{T}$ , we get  $\underline{\dim}_R(\mu) \geq \dim_{FS}(dx)$ . Since  $\mu$  was arbitrary, applying Theorem 27 for  $dx + ey \in \mathbb{T}$ , we obtain  $\dim_{FS}(dx + ey) \geq \dim_{FS}(dx)$ . The proof of  $\dim_{FS}(dx + ey) \geq \dim_{FS}(ey)$  is similar. This completes the proof of the first inequality. In order to show the second inequality, consider any  $\mu_1 \in \mathcal{W}_{dx}$ . Using Lemma 35, there exist  $\mu \in \mathcal{W}_{dx+ey}$  and  $\mu_2 \in \mathcal{W}_{ey}$  such that  $\mu = \mu_1 * \mu_2$ . Now using Lemma 32, it follows that  $\underline{\dim}_R(\mu) = \underline{\dim}_R(\mu_1 * \mu_2) \leq \underline{\dim}_R(\mu_1) + \underline{\dim}_R(\mu_2)$ . On applying Theorem 27 for the points  $dx + ey \in \mathbb{T}$  and  $ey \in \mathbb{T}$ , we get  $\dim_{FS}(dx + ey) \leq \underline{\dim}_R(\mu_1) + \text{Dim}_{FS}(ey)$ . Since  $\mu_1$  was arbitrary, applying Theorem 27 for  $dx \in \mathbb{T}$ , we obtain  $\dim_{FS}(dx + ey) \leq \dim_{FS}(dx) + \text{Dim}_{FS}(ey)$ . 2 follows similarly. ◀

The following is an immediate corollary of the Theorem 33.

► **Corollary 36.** *If  $x$  and  $y$  are real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30, then for any  $q \in \mathbb{Q}$ ,*

1.  $\dim_{FS}(x+qy) \geq \max\{\dim_{FS}(x), \dim_{FS}(y)\}$  and  $\dim_{FS}(x+qy) \leq \dim_{FS}(x) + \text{Dim}_{FS}(y)$ .
2.  $\text{Dim}_{FS}(x + qy) \geq \max\{\text{Dim}_{FS}(x), \text{Dim}_{FS}(y)\}$  and  $\text{Dim}_{FS}(x + qy) \leq \text{Dim}_{FS}(x) + \text{Dim}_{FS}(y)$

On considering the case when  $\text{Dim}_{FS}(y) = 0$ , we obtain the following corollaries, generalizing earlier results by Doty, Lutz, Nandakumar [12] and Aistleitner [1], regarding the preservation of finite-state dimension under addition with an independent sequence having zero finite-state strong dimension.

► **Corollary 37.** *If  $x$  and  $y$  are real numbers in  $\mathbb{T}$  such that  $x$  and  $y$  are independent in the sense of condition 30 with  $\text{Dim}_{FS}(y) = 0$ , then for any  $q \in \mathbb{Q}$ ,  $\dim_{FS}(x + qy) = \dim_{FS}(x)$  and  $\text{Dim}_{FS}(x + qy) = \text{Dim}_{FS}(x)$ .*

It is easy to verify that any string in  $\mathcal{C}$  is independent of any other string  $x \in \Sigma^\infty$ . Thus we obtain the following generalization of Aistleitner's result to every dimension [1].

► **Corollary 38.** *If  $y$  is any real number in  $\mathcal{C}$ , then for any  $x \in \mathbb{T}$  and  $q \in \mathbb{Q}$ ,  $\dim_{FS}(x + qy) = \dim_{FS}(x)$  and  $\text{Dim}_{FS}(x + qy) = \text{Dim}_{FS}(x)$ .*

---

## References

- 1 Christoph Aistleitner. On modifying normal numbers. *Unif. Distrib. Theory*, 6(2):49–58, 2011.
- 2 Krishna B Athreya, John M Hitchcock, Jack H Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM journal on computing*, 37(3):671–705, 2007.
- 3 Verónica Becher and Pablo Ariel Heiber. Normal numbers and finite automata. *Theoretical Computer Science*, 477:109–116, 2013.
- 4 Patrick Billingsley. *Convergence of probability measures*. John Wiley & Sons, 2013.
- 5 Chris Bourke, John M Hitchcock, and NV Vinodchandran. Entropy rates and finite-state dimension. *Theoretical Computer Science*, 349(3):392–406, 2005.

- 6 Y. Bugeaud. *Distribution Modulo 1 and Diophantine Approximation*. Pure and Applied Mathematics. Cambridge University Press, 2012.
- 7 J. W. S. Cassels. On a problem of Steinhaus about normal numbers. *Colloq. Math.*, 7:95–101, 1959. doi:10.4064/cm-7-1-95-101.
- 8 D. G. Champernowne. Construction of decimals normal in the scale of ten. *J. London Math. Soc.*, 2(8):254–260, 1933.
- 9 T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, N.Y., 1991.
- 10 Jack J Dai, James I Lathrop, Jack H Lutz, and Elvira Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310(1-3):1–33, 2004.
- 11 H. Davenport and P. Erdős. Note on normal decimals. *Canad. J. Math.*, 4:58–63, 1952. doi:10.4153/cjm-1952-005-3.
- 12 D. Doty, J. H. Lutz, and S. Nandakumar. Finite state dimension and real arithmetic. In *Proceedings of the 33rd International Colloquium on Automata, Logic and Programming*, 2006.
- 13 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.
- 14 Manfred Einsiedler and Thomas Ward. *Ergodic theory with a view towards number theory*, volume 259 of *Graduate Texts in Mathematics*. Springer-Verlag London, Ltd., London, 2011. doi:10.1007/978-0-85729-021-2.
- 15 M. Feder. Gambling using a finite state machine. *IEEE Transactions on Information Theory*, 37:1459–1461, 1991.
- 16 Gerald B Folland. *A course in abstract harmonic analysis*, volume 29. CRC press, 2016.
- 17 Michael Hochman. On self-similar sets with overlaps and inverse theorems for entropy. *Ann. of Math. (2)*, 180(2):773–822, 2014. doi:10.4007/annals.2014.180.2.7.
- 18 A. I. Khinchin. *Mathematical foundations of information theory*. Dover Publications, Inc., New York, N. Y., 1957. Translated by R. A. Silverman and M. D. Friedman.
- 19 Alexander Kozachinskiy and Alexander Shen. Two characterizations of finite-state dimension. In *International Symposium on Fundamentals of Computation Theory*, pages 80–94. Springer, 2019.
- 20 L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. Pure and Applied Mathematics. Wiley-Interscience [John Wiley & Sons], New York-London-Sydney, 1974.
- 21 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003. doi:10.1137/S0097539701417723.
- 22 Manfred G Madritsch and Bill Mance. Construction of  $\mu$ -normal sequences. *Monatshefte für Mathematik*, 179(2):259–280, 2016.
- 23 Joseph S. Miller. Extracting information is hard: a Turing degree of non-integral effective Hausdorff dimension. *Adv. Math.*, 226(1):373–384, 2011. doi:10.1016/j.aim.2010.06.024.
- 24 I. Niven. *Irrational Numbers*. Carus Mathematical Monographs, 1956.
- 25 A. Rényi. On the dimension and entropy of probability distributions. *Acta Math. Acad. Sci. Hungar.*, 10:193–215 (unbound insert), 1959. doi:10.1007/BF02063299.
- 26 Walter Rudin. *Fourier analysis on groups*. Interscience Tracts in Pure and Applied Mathematics, No. 12. Interscience Publishers (a division of John Wiley & Sons, Inc.), New York-London, 1962.
- 27 Wolfgang M. Schmidt. Über die Normalität von Zahlen zu verschiedenen Basen. *Acta Arith.*, 7:299–309, 1961/62. doi:10.4064/aa-7-3-299-309.
- 28 C. P. Schnorr and H. Stimm. Endliche Automaten und Zufallsfolgen. *Acta Informatica*, 1:345–359, 1972.
- 29 A. N. Shiryaev. *Probability*. Graduate Texts in Mathematics v.95. Springer, 2 edition, 1995.
- 30 Elias M. Stein and Rami Shakarchi. *Fourier analysis*, volume 1 of *Princeton Lectures in Analysis*. Princeton University Press, Princeton, NJ, 2003. An introduction.
- 31 D. D. Wall. *Normal Sequences*. PhD thesis, University of California, Berkeley, 1949.

## 65:16 A Weyl Criterion for Finite-State Dimension and Applications

- 32 Hermann Weyl. Über die Gleichverteilung von Zahlen mod. Eins. *Math. Ann.*, 77(3):313–352, 1916. doi:10.1007/BF01475864.
- 33 Lai-Sang Young. Dimension, entropy and Lyapunov exponents. *Ergodic theory and dynamical systems*, 2(1):109–124, 1982.
- 34 J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transaction on Information Theory*, 24:530–536, 1978.

# On the Complexity Dichotomy for the Satisfiability of Systems of Term Equations over Finite Algebras

Peter Mayr  

Department of Mathematics, University of Colorado Boulder, CO, USA  
Institute for Algebra, Johannes Kepler Universität Linz, Austria

---

## Abstract

For a fixed finite algebra  $\mathbf{A}$ , we consider the decision problem  $\text{SysTerm}(\mathbf{A})$ : does a given system of term equations have a solution in  $\mathbf{A}$ ? This is equivalent to a constraint satisfaction problem (CSP) for a relational structure whose relations are the graphs of the basic operations of  $\mathbf{A}$ . From the complexity dichotomy for CSP over fixed finite templates due to Bulatov [4] and Zhuk [18], it follows that  $\text{SysTerm}(\mathbf{A})$  for a finite algebra  $\mathbf{A}$  is in P if  $\mathbf{A}$  has a not necessarily idempotent Taylor polymorphism and is NP-complete otherwise. More explicitly, we show that for a finite algebra  $\mathbf{A}$  in a congruence modular variety (e.g. for a quasigroup),  $\text{SysTerm}(\mathbf{A})$  is in P if the core of  $\mathbf{A}$  is abelian and is NP-complete otherwise. Given  $\mathbf{A}$  by the graphs of its basic operations, we show that this condition for tractability can be decided in quasi-polynomial time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Problems, reductions and completeness

**Keywords and phrases** systems of equations, general algebras, constraint satisfaction

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.66

**Funding** Partially supported by the Austrian Science Fund (FWF): P33878.

**Acknowledgements** I want to thank E. Aichinger for discussions on this problem and the referees for their diligent reading and comments.

## 1 Introduction

How hard is it to check whether a system of term equations is solvable in an algebra? The *System of Term Equations Satisfiability Problem* over a fixed algebra  $\mathbf{A}$  is the following decision problem:

$\text{SysTerm}(\mathbf{A})$   
Input: terms  $s_1, t_1, \dots, s_m, t_m$  in the signature of  $\mathbf{A}$   
Problem: Does  $s_1 \approx t_1, \dots, s_m \approx t_m$  have a solution in  $\mathbf{A}$ ?

For example,  $\text{SysTerm}$  for the ring of integers  $(\mathbb{Z}, +, \cdot, 1)$  is Hilbert's tenth problem and undecidable by Matiyasevich's theorem. In this note we only consider  $\text{SysTerm}$  for finite algebras (meaning algebraic structures of finite size and finite signature), which clearly can be solved in non-deterministic polynomial time (NP).

Obviously  $\text{SysTerm}(\mathbf{A})$  has always a positive answer if  $\mathbf{A}$  has a trivial subalgebra  $\{o\}$  by setting all variables to  $o$ . Hence it is trivial for most classical algebras like groups, rings (without 1 as basic operation), lattices, and semigroups with idempotents<sup>1</sup>. This is one reason why the related problem  $\text{SysPol}$ , the satisfiability problem for a system of polynomial equations (allowing also constants), has received more attention in the past. Note that  $\text{SysPol}$  can be considered as the restriction of  $\text{SysTerm}$  to algebras for which each constant

---

<sup>1</sup> Rings with 1, quasigroups, more generally magmas, and  $G$ -sets are some of the few named algebras without trivial subalgebras. However, among finite algebras with randomly chosen operations, almost none have trivial subalgebras by a result of Murskii [2, Theorem 6.16].



© Peter Mayr;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 66; pp. 66:1–66:12

Leibniz International Proceedings in Informatics



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

is a basic operation. We will show that for any finite  $\mathbf{A}$  there exists some algebra  $\mathbf{A}'$  (the core of  $\mathbf{A}$ ) such that  $\text{SysTerm}(\mathbf{A})$  and  $\text{SysPol}(\mathbf{A}')$  are equivalent under logspace reductions (see Lemmas 6 and 7).

Goldmann and Russell [11] showed that  $\text{SysPol}$  (which they denote as  $\text{EQN}^*$ ) is in  $\text{P}$  for abelian and  $\text{NP}$ -complete for non-abelian finite groups. Klíma, Tesson and Thérien [14] investigated  $\text{SysPol}$  over finite semigroups and showed that it is in  $\text{P}$  for commutative monoids that are unions of their subgroups and  $\text{NP}$ -complete for other monoids. Larose and Zádori [15] studied  $\text{SysPol}$  over finite algebras in general and observed that they are logspace-equivalent to constraint satisfaction problems (CSP) of a specific form. They showed in particular that  $\text{SysPol}$  for any finite  $\mathbf{A}$  in a congruence modular variety (including most classical algebras like groups, modules, rings, quasigroups, lattices but not semigroups) is in  $\text{P}$  if  $\mathbf{A}$  is abelian (meaning the operations of  $\mathbf{A}$  are affine functions over an abelian group) and  $\text{NP}$ -complete otherwise. By using the universal algebraic definition of commutators and abelianness, this generalizes the previously mentioned result of Goldmann and Russell. Broniek [3] investigated  $\text{SysPol}$  and  $\text{SysTerm}$  for unary algebras (where all basic operations are unary). He showed in particular that  $\text{SysTerm}$  for unary algebras of size at most 3 is in  $\text{P}$  or  $\text{NP}$ -complete.

Our goal in this note is first to explicitly state the connections between  $\text{SysTerm}$ ,  $\text{SysPol}$  and  $\text{CSP}$  by a straightforward adaptation of the approach of Larose and Zádori in Section 3. From the celebrated complexity dichotomy for  $\text{CSP}$  by Bulatov [4] and Zhuk [18], we then obtain immediately that  $\text{SysTerm}$  for any finite algebra is either in  $\text{P}$  or  $\text{NP}$ -complete in Theorem 1. For finite  $\mathbf{A}$  in a congruence modular variety, we give an algebraic criterion for when  $\text{SysTerm}(\mathbf{A})$  is tractable in Theorem 2. Finally we show that this criterion can be decided in quasi-polynomial time for  $\mathbf{A}$  given by the graphs of its operations in Theorem 3.

For the precise statement of our results we recall some notions that play an important role in the classification of  $\text{CSP}$ s and algebras. For a structure  $\mathcal{C}$  (possibly with function and relation symbols) define *polymorphisms* of  $\mathcal{C}$  as the homomorphisms from finite powers of  $\mathcal{C}$  to  $\mathcal{C}$  and denote the set of polymorphisms as

$$\text{Pol}(\mathcal{C}) := \bigcup_{n \in \mathbb{N}} \text{Hom}(\mathcal{C}^n, \mathcal{C}).$$

For example, the polymorphisms of a vector space  $\mathbf{A}$  are just the linear maps from  $\mathbf{A}^n$  to  $\mathbf{A}$  for  $n \in \mathbb{N}$ .

Let  $f: A^n \rightarrow A$  for  $n > 1$ . Then  $f$  is *Taylor* if it satisfies  $n$  identities in distinct variables  $x, y$  of the form

$$f(\dots, x_i, \dots) \approx f(\dots, y_i, \dots) \text{ for all } i \in \{1, \dots, n\}$$

where the omitted variables on either side may be  $x$  or  $y$ . These identities were chosen so that no projection map on a non-trivial domain can satisfy them.

Next  $f: A^4 \rightarrow A$  is *Siggers* if it satisfies

$$f(a, r, e, a) \approx f(r, a, r, e).$$

For example, a binary commutative operation  $f$  is Taylor by virtue of  $f(x, y) \approx f(y, x)$ . By adding two fictitious variables we also obtain a Siggers operation from  $f$ . Clearly every Siggers operation is Taylor. In fact, every finite structure  $\mathcal{C}$  has a Taylor polymorphism of some arity iff  $\mathcal{C}$  has a Siggers polymorphism [13, 17].

As in [1] we do not require that Taylor and Siggers operations are idempotent like in older literature (see Lemma 8 for the relation with their idempotent version). This allows for a convenient formulation of the dichotomy for  $\text{CSP}$  and consequently the dichotomy for  $\text{SysTerm}$ .

► **Theorem 1.** *Let  $\mathbf{A}$  be a finite algebra. Then  $\text{SysTerm}(\mathbf{A})$  is in P if  $\mathbf{A}$  has a (not necessarily idempotent) Taylor (equivalently Siggers) polymorphism; else  $\text{SysTerm}(\mathbf{A})$  is NP-complete.*

We will show Theorem 1 in Section 3 by encoding  $\text{SysTerm}$  over an algebra as CSP over a relational structure and invoking the dichotomy for CSPs.

We already observed that  $\text{SysTerm}$  for  $\mathbf{A}$  with a trivial subalgebra  $\{o\}$  is trivial. Still to put this into the context of Theorem 1, note that such an algebra has  $f(x, y) := o$  as a Taylor polymorphism. For a less obvious example,  $\mathbf{A} = (\mathbb{Z}_2, +, 0, 1)$  has  $x + y + z$  as Taylor polymorphism and consequently tractable  $\text{SysTerm}$ .

Theorem 1 generalizes all the dichotomy results for  $\text{SysPol}$  in [11, 14, 15] mentioned above and fully settles the P/NP-complete dichotomy for  $\text{SysTerm}$ . Still it would be desirable to describe its boundary in more explicit structural terms of the algebra  $\mathbf{A}$  than by the existence of certain polymorphisms. We manage to do this under the assumption of additional structural properties on  $\mathbf{A}$ .

Here we just review the bare minimum of notions from universal algebra that we need to state our results. For more details we refer to [2, 5, 16] and Section 2 below. A *variety* is a class of algebras of fixed signature that is defined by identities. For example, groups form a variety with a binary operation  $\cdot$ , a unary  $^{-1}$  and constant 1 satisfying the usual group axioms. Varieties are usually classified by so-called Mal'cev conditions, essentially the term identities they satisfy. We list the conditions which occur in this note in increasing strength.

- A variety  $V$  is *Taylor* if it has a term  $t$  which induces an idempotent Taylor operation on all its algebras. Here idempotent means that  $V$  satisfies  $t(x, \dots, x) \approx x$ .

For example, semilattices form a Taylor variety with Taylor term  $t(x, y) := xy$  but (commutative) semigroups and  $G$ -sets do not.

- A variety  $V$  is *congruence modular* if every algebra  $\mathbf{A}$  in  $V$  has a modular congruence lattice.

Most classical algebras, in particular those that have (quasi)group operations, like groups, modules, rings, loops, ... or lattice operations, like lattices, Boolean algebras, Heyting algebras, ... are members of congruence modular varieties. On the other hand, semilattices and more generally semigroups do not form congruence modular varieties.

- A variety is *congruence distributive* if all its algebras have distributive congruence lattices. Every algebra with lattice operations is contained in a congruence distributive variety but non-trivial groups are not.

Since distributivity implies modularity for lattices, congruence distributive varieties are congruence modular. Further congruence modular varieties are Taylor.

There exist various generalizations of commutators from groups to arbitrary algebras. These may differ in general but most of them lead to the same concept of abelianness in Taylor varieties (see [9, 12]). In particular, a finite algebra  $\mathbf{A}$  in a Taylor variety is *abelian* (with respect to the standard term condition commutator) iff all its basic operations are affine functions of some commutative group  $(A, +, -, 0)$ . Although such an abelian algebra  $\mathbf{A}$  may not have  $+$  or  $-$  as term operations, the ternary function  $x - y + z$  is a term operation and is called the *Mal'cev term operation* of  $\mathbf{A}$ . A group is abelian in this sense iff it is abelian in the classical group theoretic sense. A loop is abelian iff it is an abelian group. Since for a ring the commutator of congruences corresponds to the product of ideals, a ring is abelian iff its multiplication is 0. For a lattice or any algebra in a congruence distributive variety, the commutator of two congruences is just their intersection. Hence these algebras are abelian iff they are trivial.

Next we extend some established notions and facts on relational structures to the setting of algebras. A finite structure  $\mathcal{C}$  (possibly with function and relational symbols) is a *core* if every endomorphism of  $\mathcal{C}$  is an embedding (equivalently, an automorphism). It is well-known

and not hard to see that if  $h$  is an endomorphism of a finite structure  $\mathcal{C}$  such that  $h(\mathcal{C})$  is minimal with respect to inclusion among all endomorphic images of  $\mathcal{C}$ , then  $h(\mathcal{C})$  is a core. Moreover this core is unique up to isomorphism and hence called *the core of  $\mathcal{C}$* . An algebra has trivial core iff it has a trivial subalgebra. An algebra expanded with all constants is its own core. For a non-degenerate example, the core of the symmetric group  $(S_3, \cdot, (1, 2))$  expanded with the additional constant  $(1, 2)$  is isomorphic to  $(\mathbb{Z}_2, +, 0, 1)$ .

We will prove the following generalization of a result by Larose and Zádori on the complexity of systems of polynomial equations [15, Corollary 3.14] in Section 4.

► **Theorem 2.** *Let  $\mathbf{A}$  be a finite algebra in a congruence modular variety. Then  $\text{SysTerm}(\mathbf{A})$  is in P if the core of  $\mathbf{A}$  is abelian; else  $\text{SysTerm}(\mathbf{A})$  is NP-complete.*

Since any non-trivial ring with 1 is non-abelian by the discussion of commutators above, it follows that its  $\text{SysTerm}$  is NP-complete.

Also, since non-trivial algebras in congruence distributive varieties are non-abelian, Theorem 2 yields that  $\text{SysTerm}(\mathbf{A})$  for such a finite  $\mathbf{A}$  is NP-complete unless  $\mathbf{A}$  has a trivial subalgebra and hence trivial core, in which case  $\text{SysTerm}(\mathbf{A})$  is trivial.

A natural follow-up to the dichotomy results above is the metaquestion of deciding for a given algebra  $\mathbf{A}$  whether it has tractable  $\text{SysTerm}$ . Or, for practical purposes, how much preprocessing is necessary on a given algebra  $\mathbf{A}$  with abelian core such that one can solve  $\text{SysTerm}$  for  $\mathbf{A}$  in polynomial time in the size of the system of equations? Here and in the following we assume that algebras are given by the graphs of their basic operations.

Recall that a finite structure has a (not necessarily idempotent) Taylor polymorphism iff it has a (not necessarily idempotent) 4-ary Siggers polymorphism. The latter condition can clearly be decided in NP. So the metaquestion for  $\text{SysTerm}$ , i.e., deciding whether a given finite algebra  $\mathbf{A}$  has a (not necessarily idempotent) Taylor polymorphism, is in NP.

Chen and Larose showed that the metaquestion for CSP, i.e., deciding whether a given finite relational structure  $\mathbb{A}$  has a (not necessarily idempotent) Taylor polymorphism, is actually NP-complete [6]. Klíma, Tesson and Thérien constructed for every finite relational structure  $\mathbb{A}$  a finite semigroup  $\mathbf{A}$  such that  $\text{CSP}(\mathbb{A})$  is polynomial time equivalent to  $\text{SysPol}(\mathbf{A})$  [14, Theorem 8]. Similarly, Broniek constructed for every finite relational structure  $\mathbb{A}$  a finite unary algebra  $\mathbf{A}$  such that  $\text{CSP}(\mathbb{A})$  is polynomial time equivalent to  $\text{SysTerm}(\mathbf{A})$  [3, Theorem 3.4]. However, for both these constructions, the size of the algebra  $\mathbf{A}$  is exponential in the size of relational structure  $\mathbb{A}$ . Hence they do not allow to transfer the NP-hardness of the metaquestion for CSP to the metaquestion for  $\text{SysTerm}$ . To the best of our knowledge, it may be easier to decide for algebras whether they have a Taylor polymorphism than for relational structures.

In particular, for an algebra in a congruence modular variety, the existence of a (not necessarily idempotent) Taylor polymorphism can be decided in quasi-polynomial time by the following slightly stronger result, which we will prove in Section 5. Recall that congruence modular varieties are Taylor.

► **Theorem 3.** *There exists a quasi-polynomial time algorithm that, given a finite algebra  $\mathbf{A}$  in a Taylor variety, decides if the core of  $\mathbf{A}$  is abelian, in which case the core of  $\mathbf{A}$  and the graph of its Mal'cev term operation can also be computed in quasi-polynomial time.*

Thus given a finite algebra  $\mathbf{A}$  in a Taylor variety, we can compute its core  $\mathbf{C}$  in quasi-polynomial time if  $\mathbf{C}$  is abelian. Moreover, if the core  $\mathbf{C}$  is abelian, we can use its Mal'cev term operation  $x - y + z$  to reduce  $\text{SysTerm}(\mathbf{A})$  to a linear system of equations over the abelian group  $(C, +)$ . If a system of term equations over abelian  $\mathbf{C}$  (equivalently over  $\mathbf{A}$ ) has a solution, then we can also find it in polynomial time in the size of the system.

We do not know whether the quasi-polynomial time algorithms in Theorem 3 can be improved to polynomial time.

Of course one can also ask how hard it is to check the assumptions of Theorem 3 in case that the idempotent Taylor term operation of  $\mathbf{A}$  is not given as part of the input. For that Freese and Valeriote showed that deciding whether a given finite algebra  $\mathbf{A}$  is in a Taylor variety or whether it is in a congruence modular variety is EXPTIME-complete [10, Corollary 9.3] but that these problems are in P for idempotent  $\mathbf{A}$ , i.e., if all basic operations  $f$  of  $\mathbf{A}$  satisfy  $f(x, \dots, x) \approx x$  [10, Theorem 6.2, 6.3].

## 2 Preliminaries

We review the algebraic results that we will need. For standard universal algebraic background we refer the reader to [2, 5, 9, 12, 16].

### 2.1 Algebras and varieties

An *algebra*  $\mathbf{A} = (A, \{f^{\mathbf{A}} : f \in F\})$  is a pair where  $A$  is a non-empty set (the *universe* of  $\mathbf{A}$ ),  $F$  is a set of function symbols equipped with a map *arity*:  $F \rightarrow \mathbb{N}$  that assigns to each function symbol its arity (the *signature* of  $\mathbf{A}$ ), and  $f^{\mathbf{A}}$  are the interpretations of the symbols  $f \in F$  as operations of the corresponding arity on  $A$  (the *basic operations* of  $\mathbf{A}$ ). We say  $\mathbf{A}$  is *finite* if its universe and its signature are finite. An algebra is *trivial* if its universe has size 1.

*F-terms* or *terms in the signature* of  $\mathbf{A}$  are constructed from function symbols  $F$  and variables  $x_1, x_2, \dots$  in the usual way: every variable is an  $F$ -term, and if  $f \in F$  is  $k$ -ary and  $t_1, \dots, t_k$  are  $F$ -terms, then  $f(t_1, \dots, t_k)$  is an  $F$ -term. Every  $F$ -term  $t$  in variables  $x_1, \dots, x_k$  induces a  $k$ -ary *term function*  $t^{\mathbf{A}}: A^k \rightarrow A$  by interpreting a variable  $x_i$  as the  $i$ -th projection from  $A^k$  onto  $A$  and interpreting function symbols  $f$  as  $f^{\mathbf{A}}$ .

*Polynomials* over  $\mathbf{A}$  are defined like terms except that additionally for every element  $a \in A$  there is a constant polynomial  $a$ . Again every polynomial  $p$  in variables  $x_1, \dots, x_k$  induces a  $k$ -ary *polynomial function*  $p^{\mathbf{A}}: A^k \rightarrow A$  via the interpretation of function symbols in  $F$  on  $A$  and the interpretation of a constant  $a$  as the corresponding element  $a \in A$ . Two algebras  $\mathbf{A}_1 = (A, F_1)$  and  $\mathbf{A}_2 = (A, F_2)$  on the same universe are *polynomially equivalent* if  $\mathbf{A}_1$  and  $\mathbf{A}_2$  have the same set of polynomial functions of all arities.

An *identity* is a pair of terms  $(s, t)$ , which we usually write as  $s \approx t$ . For  $k$ -ary terms  $s, t$ , the  $k$ -tuple  $(a_1, \dots, a_k) \in A^k$  is a *solution* of  $s(x_1, \dots, x_k) \approx t(x_1, \dots, x_k)$  if  $s^{\mathbf{A}}(a_1, \dots, a_k) = t^{\mathbf{A}}(a_1, \dots, a_k)$ . An algebra  $\mathbf{A}$  *satisfies* an identity  $s \approx t$  if  $s^{\mathbf{A}} = t^{\mathbf{A}}$ .

A *variety*  $V$  is a class of algebras over the same fixed signature  $F$  that is defined by a set of identities  $\Sigma$ , that is,  $V = \{\mathbf{A} : \mathbf{A} \text{ satisfies } \Sigma\}$ . Birkhoff showed that the variety generated by a class  $K$  of algebras over  $F$  consists of all homomorphic images of subalgebras of direct powers of elements in  $K$ .

A variety  $V$  is *locally finite* if all its finitely generated algebras are finite. For example, the variety generated by a finite algebra  $\mathbf{A}$  is locally finite.

### 2.2 Commutators

Commutators have been generalized from normal subgroups of groups to congruences of general algebras by Smith, Hagemann, Herrmann, Gumm, Freese, McKenzie and others. See [9] for the history and overview of their development.

For congruences  $\alpha, \beta$  of an algebra  $\mathbf{A}$ , let  $M_{\mathbf{A}}(\alpha, \beta)$  be the subalgebra of  $\mathbf{A}^{2 \times 2}$  that is generated by all elements of the form

$$\begin{bmatrix} a & a \\ b & b \end{bmatrix}, \begin{bmatrix} c & d \\ c & d \end{bmatrix} \quad \text{for } a\alpha b, c\beta d.$$

Writing quadruples as  $2 \times 2$  tables is just a notational convenience here. The operations of  $\mathbf{A}$  simply apply componentwise. The *commutator*  $[\alpha, \beta]$  is the smallest congruence  $\gamma$  of  $\mathbf{A}$  such that

$$\forall \begin{bmatrix} x & y \\ z & w \end{bmatrix} \in M_{\mathbf{A}}(\alpha, \beta): x\gamma y \Rightarrow z\gamma w.$$

For an algebra  $\mathbf{A}$  let  $1_A$  denote the total congruence and  $0_A$  denote the trivial congruence (equality). Then  $\mathbf{A}$  is *abelian* if  $[1_A, 1_A] = 0_A$ .

Abelianess has strong structural consequences of an algebra. Assume  $\mathbf{A}$  is a finite abelian algebra in a Taylor variety. Then  $\mathbf{A}$  is polynomially equivalent to a module by [12, 9]. More explicitly, there exist operation  $+, -, 0$  such that  $(A, +, -, 0)$  is an abelian group with a set of endomorphisms  $R$ . Every  $k$ -ary basic operation  $f^{\mathbf{A}}$  of  $\mathbf{A}$  can be represented in the form

$$f^{\mathbf{A}}(x_1, \dots, x_k) = \sum_{i=1}^k r_i(x_i) + c$$

for some endomorphisms  $r_1, \dots, r_k \in R$  and some constant  $c \in A$ . Given a term  $t$  of  $\mathbf{A}$  it can be rewritten iteratively as a sum of endomorphisms of  $(A, +, -, 0)$  and constants in polynomial time as well. Hence  $\text{SysTerm}(\mathbf{A})$  reduces to solving a system of linear equations over  $(A, +, -, 0)$ , which is clearly in P.

### 2.3 Tame congruence theory

For congruence  $\alpha, \beta$  of an algebra  $\mathbf{A}$  we say  $\alpha$  is *covered* by  $\beta$  (written  $\alpha \prec \beta$ ) if  $\alpha$  is strictly contained in  $\beta$  and there is no congruence strictly between  $\alpha$  and  $\beta$ .

A finite non-trivial algebra is *minimal* if all its unary polynomial operations are either constant or permutations. Pálffy showed that every minimal algebra is polynomially equivalent to an algebra of one the following five types:

1. a  $G$ -set (i.e., an algebra all of whose basic operations are permutations);
2. a vector space;
3. the Boolean algebra of size 2;
4. the lattice of size 2;
5. the semilattice of size 2.

Tame congruence theory (TCT) as developed by Hobby and McKenzie in [12] associates with any pair of congruences  $\alpha \prec \beta$  of a finite algebra  $\mathbf{A}$  a set of minimal algebras all of which have the same type 1-5. The precise construction is quite technical and will not be needed in this paper. Hence we will not discuss it beyond stating that every pair  $\alpha \prec \beta$  is labelled with a unique type. The set of all types of pairs  $\alpha \prec \beta$  of  $\mathbf{A}$  is denoted by  $\text{typ}\{\mathbf{A}\}$ . For a variety  $V$  the set of all types of  $\alpha \prec \beta$  of all finite algebras  $\mathbf{A}$  in  $V$  is denoted by  $\text{typ}\{V\}$ .

There are deep connections between the typeset of a variety  $V$ , that is, the local behaviour of polynomial functions on its finite members, and the identities that hold in  $V$ . For example:

- [12, Theorem 9.6] A locally finite variety  $V$  is Taylor iff  $1 \notin \text{typ}\{V\}$ .
- [12, Theorem 8.5] If a locally finite variety  $V$  is congruence modular, then  $\text{typ}\{V\} \subseteq \{2, 3, 4\}$ .

### 3 SysTerm, SysPol and CSP

In this section we collect easy facts on the correspondence between systems of equations and constraint satisfaction problems that may be known at least implicitly in one form or the other. Still we hope it is useful to provide an explicit and consistent overview for the reader.

#### 3.1 Reduction to CSP

First we reduce a system of equations to a particular constraint satisfaction problem. The *Constraint Satisfaction Problem* over a fixed relational structure  $\mathbb{A}$  is the decision problem:

CSP( $\mathbb{A}$ )  
 Input: a structure  $\mathbb{X}$  in the signature of  $\mathbb{A}$   
 Problem: Is there a homomorphism from  $\mathbb{X}$  to  $\mathbb{A}$ ?

Many classical decision problems like 3-SAT, graph coloring, solvability of linear systems... can be formulated as CSP for an appropriately chosen structure  $\mathbb{A}$ . For background on CSP on fixed templates we refer to the survey [1] by Barto, Krokhn and Willard.

Denote the *graph* of a  $k$ -ary operation  $f: A^k \rightarrow A$  by the  $k+1$ -ary relation

$$f^\circ := \{(x_1, \dots, x_k, f(x_1, \dots, x_k)) : x_1, \dots, x_k \in A\}.$$

For an algebra  $\mathbf{A} = (A, F)$  with universe  $A$  and basic operations  $F$ , let  $\mathbb{A}^\circ := (A, \{f^\circ : f \in F\})$  denote the relational structure with the graphs of the basic operations as relations.

Larose and Zádori observed the following correspondence between SysPol and CSP. The proof for systems of term equations is essentially the same and added here for the convenience of the reader.

► **Lemma 4** ([15, cf. Theorem 2.2]). *Let  $\mathbf{A}$  be a finite algebra. Then  $\text{SysTerm}(\mathbf{A})$  is logspace-equivalent to  $\text{CSP}(\mathbb{A}^\circ)$ .*

**Proof.** For a  $\text{CSP}(\mathbb{A}^\circ)$ -instance  $\mathbb{X}$ , each constraint  $(x_{i_1}, \dots, x_{i_{k+1}}) \in f^\circ$  can be reformulated as  $f(x_{i_1}, \dots, x_{i_k}) \approx x_{i_{k+1}}$  in constant time. Clearly the conjunction of constraints is satisfiable iff the system of corresponding equations is solvable in  $\mathbf{A}$ .

Conversely, for a  $\text{SysTerm}(\mathbf{A})$ -instance  $s_1 \approx t_1, \dots, s_m \approx t_m$ , rewrite each occurring term  $t = f(u_1, \dots, u_k)$  for a basic operation  $f$  of  $\mathbf{A}$  as a sequence of constraints  $(y_{u_1}, \dots, y_{u_k}, y_t) \in f^\circ$  in variables  $y$  indexed by subterms and correspondingly for the subterms  $u_1, \dots, u_k$ . If  $t = x$  is a variable, just write  $y_t = x$ . All these constraints together with  $y_s = y_t$  for every given equation  $s \approx t$  form a  $\text{CSP}(\mathbb{A}^\circ)$ -instance which is satisfiable iff the original system of equations over  $\mathbf{A}$  is solvable. This rewriting creates as many new variables as there are function symbols and variables in  $s_1, t_1, \dots, s_m, t_m$  and can be done in logarithmic space. ◀

It is straightforward to check that the polymorphisms of  $\mathbf{A}$  are the same as those of  $\mathbb{A}^\circ$ .

► **Lemma 5.**  $\text{Pol}(\mathbf{A}) = \text{Pol}(\mathbb{A}^\circ)$  for every algebra  $\mathbf{A}$ .

**Proof.** Let  $h: A^n \rightarrow A$ ,  $f: A^k \rightarrow A$  and  $x_1 = (x_{11}, \dots, x_{1n}), \dots, x_k = (x_{k1}, \dots, x_{kn})$  in  $A^n$ . Then  $hf(x_1, \dots, x_k) = f(h(x_1), \dots, h(x_k))$  with  $f$  acting on  $A^n$  componentwise iff

$$h \left( \left[ \begin{array}{c} x_{11} \\ \vdots \\ x_{k1} \\ f(x_{11}, \dots, x_{k1}) \end{array} \right], \dots, \left[ \begin{array}{c} x_{1n} \\ \vdots \\ x_{kn} \\ f(x_{1n}, \dots, x_{kn}) \end{array} \right] \right) \in f^\circ.$$

Hence  $h$  is a polymorphism of the algebra  $(A, f)$  iff  $h$  is a polymorphism of the relational structure  $(A, f^\circ)$ . The assertion follows. ◀

The previous two lemmas are already enough to obtain the complexity dichotomy for SysTerm from that for CSP.

**Proof of Theorem 1.** By Lemma 4 and 5 it suffices to consider  $\text{CSP}(\mathbb{A}^\circ)$ . Then the hardness part only uses that 3-SAT reduces to  $\text{CSP}(\mathbb{A}^\circ)$  if  $\mathbb{A}^\circ$  has no Taylor polymorphism [1, Theorem 40].

The tractability part follows from the celebrated result by Bulatov [4] and Zhuk [18] that  $\text{CSP}(\mathbb{A})$  for a finite relational structure  $\mathbb{A}$  is in P if  $\mathbb{A}$  has a (not necessarily idempotent) Taylor polymorphism. ◀

### 3.2 Cores

As for CSP, it suffices to investigate SysTerm for cores  $\mathbf{A}$  by the next observation.

► **Lemma 6.**  $\text{SysTerm}(\mathbf{A}) = \text{SysTerm}(h(\mathbf{A}))$  for each endomorphism  $h$  of  $\mathbf{A}$ .

**Proof.** Let  $h$  be an endomorphism of  $\mathbf{A}$ . Obviously, if a system of term identities  $\Sigma$  has a solution in the subalgebra  $h(\mathbf{A})$  of  $\mathbf{A}$ , then also in  $\mathbf{A}$ . Conversely, if  $\Sigma$  has a solution in  $\mathbf{A}$ , then clearly also in its homomorphic image  $h(\mathbf{A})$ . ◀

It is well-known that a CSP over a core relational structure is equivalent to the CSP over its expansion with singletons. Correspondingly, systems of term equations over a core algebra  $\mathbf{A}$  are equivalent to systems of polynomial equations over  $\mathbf{A}$ . We give a direct proof of this fact since it is short and makes the reduction from SysPol to SysTerm more apparent.

► **Lemma 7.** Let  $\mathbf{A}$  be a finite algebra that is a core. Then  $\text{SysPol}(\mathbf{A})$  is logspace-equivalent to  $\text{SysTerm}(\mathbf{A})$ .

**Proof.**  $\text{SysTerm}(\mathbf{A})$  reduces trivially to  $\text{SysPol}(\mathbf{A})$ . For the converse, the crucial observation is that the graphs of endomorphisms of  $\mathbf{A}$  are the solutions of a system of term equations. By definition a map  $h: A \rightarrow A$  is an endomorphism of  $\mathbf{A}$  iff for all  $f \in F$ , say  $k$ -ary, and for all  $a_1, \dots, a_k \in A$

$$f(h(a_1), \dots, h(a_k)) = h(f(a_1, \dots, a_k)).$$

Hence  $\{(a, h(a)) : a \in A\}$  is the graph of an endomorphism of  $\mathbf{A}$  iff  $y_a = h(a)$  for  $a \in A$  is a solution of the system of term equations

$$f(y_{a_1}, \dots, y_{a_k}) \approx y_{f(a_1, \dots, a_k)} \quad \text{for } f \in F \text{ (} k\text{-ary), } a_1, \dots, a_k \in A. \quad (1)$$

Given an instance of  $\text{SysPol}(\mathbf{A})$  with variables  $x_1, \dots, x_n$ , we introduce  $|A|$  new variables  $y_a$  for  $a \in A$  and replace every occurrence of a constant  $a$  in a polynomial equation by  $y_a$ . To the resulting set of term equations we also add the equations (1) to obtain an instance of  $\text{SysTerm}(\mathbf{A})$ . Note that the added system (1) does not depend on the original input, only on  $\mathbf{A}$ . Hence the new term system can be obtained from the original polynomial system in logspace by rewriting any occurring constant  $a$  as variable  $y_a$ .

If the original polynomial system has a solution, then clearly the new term system has a solution with  $y_a = a$ . Conversely, if the new system has a solution  $x_1 = b_1, \dots, x_n = b_n$  and  $y_a = h(a)$  for  $a \in A$ , then  $h$  is an endomorphism of  $\mathbf{A}$ . Since  $\mathbf{A}$  is a core by assumption,  $h$  is in fact an automorphism. Hence  $x_1 = h^{-1}(b_1), \dots, x_n = h^{-1}(b_n)$  and  $y_a = a$  for  $a \in A$  is also a solution of the new system. Thus  $x_1 = h^{-1}(b_1), \dots, x_n = h^{-1}(b_n)$  is a solution of the original polynomial system. ◀

### 3.3 Polymorphisms satisfying height-1 identities

An identity has *height* 1 if it is of the form  $f(u_1, \dots, u_k) \approx g(v_1, \dots, v_\ell)$  for operation symbols  $f, g$  and not necessarily distinct variables  $u_1, \dots, u_k, v_1, \dots, v_\ell$ .

For example, commutativity of a binary operation  $f$  is expressed by a height-1 identity but associativity is not since that requires nested applications of  $f$ .

Since the identities for a Taylor operation all have height 1, the next lemma yields that a structure has a (not necessarily idempotent) Taylor polymorphism iff its core has the corresponding idempotent polymorphism. For relational structures this is well-known and the same easy proof applies to general structures.

► **Lemma 8** ([6, cf. Lemma 6.4]). *Let  $\Sigma$  be a set of height-1 identities. Then a finite structure  $\mathcal{C}$  (possibly with function and relation symbols) has polymorphisms satisfying  $\Sigma$  iff the core of  $\mathcal{C}$  has idempotent polymorphisms satisfying  $\Sigma$ .*

**Proof.** Let  $h$  be an endomorphism of  $\mathcal{C}$  such that  $h(\mathcal{C})$  is the core of  $\mathcal{C}$ .

If  $F$  is a set of polymorphisms of  $h(\mathcal{C})$  satisfying  $\Sigma$ , then  $f'(x_1, \dots, x_k) := f(h(x_1), \dots, h(x_k))$  for  $f \in F$  ( $k$ -ary) are polymorphisms of  $\mathcal{C}$  and still satisfy the same height-1 identities.

Conversely, let  $F$  be polymorphisms of  $\mathcal{C}$  satisfying  $\Sigma$ . For  $f \in F$  ( $k$ -ary), let  $f^*(x_1, \dots, x_k)$  be the restriction of  $hf(x_1, \dots, x_k)$  to the substructure  $h(\mathcal{C})$ . Then  $\{f^* : f \in F\}$  is a set of polymorphism of  $h(\mathcal{C})$  which still satisfies the height-1 identities of  $\Sigma$ . Moreover, for every  $f \in F$ , we have that  $f_1(x) := f^*(x, \dots, x)$  is an endomorphism of  $h(\mathcal{C})$ , hence an automorphism. Thus  $f_1^{-1}f^*$  is an idempotent polymorphism of  $h(\mathcal{C})$ . If  $f^*(u_1, \dots, u_k) \approx g^*(v_1, \dots, v_\ell)$  is in  $\Sigma$ , then  $f_1 = g_1$  and consequently  $f_1^{-1}f^*(u_1, \dots, u_k) \approx g_1^{-1}g^*(v_1, \dots, v_\ell)$  holds on  $h(\mathcal{C})$ . Hence  $f_1^{-1}f^*$  for  $f \in F$  are idempotent polymorphisms of  $h(\mathcal{C})$  that still satisfy  $\Sigma$ . ◀

## 4 Systems over algebras in congruence modular varieties

Larose and Zádori explicitly characterized finite algebras without congruences  $\alpha \prec \beta$  of TCT type 5 in Taylor varieties that have idempotent Taylor polymorphisms.

► **Theorem 9** ([15, Theorem 3.12]). *Let  $\mathbf{A}$  be a finite algebra in a Taylor variety such that  $5 \notin \text{typ}\{\mathbf{A}\}$ . Then  $\mathbf{A}$  has an idempotent Taylor polymorphism iff  $\mathbf{A}$  is abelian.*

Note that there exist non-abelian algebras with idempotent Taylor term operations that commute with themselves, e.g., semilattices. Hence the assumption  $5 \notin \text{typ}\{\mathbf{A}\}$  cannot be omitted in Theorem 9.

Theorem 9 yields an explicit characterization of the complexity of SysTerm over cores that parallels those of SysPol by Larose and Zádori.

► **Corollary 10** ([15, cf. Corollary 3.13]). *Let  $\mathbf{A}$  be a finite algebra in a Taylor variety such that  $5 \notin \text{typ}\{\mathbf{A}\}$ . Then  $\text{SysTerm}(\mathbf{A})$  is in P if the core of  $\mathbf{A}$  is abelian; else  $\text{SysTerm}(\mathbf{A})$  is NP-complete.*

**Proof.** If the core  $h(\mathbf{A})$  of  $\mathbf{A}$  is abelian, then  $h(\mathbf{A})$  is polynomially equivalent to a module with group operations  $+, -$  by [12]. Further  $\text{SysTerm}(h(\mathbf{A}))$  reduces to a system of linear equations over that module. Then  $d(x, y, z) = x - y + z$  is a polymorphism of  $h(\mathbf{A})$  and also of the corresponding relational structure  $h(\mathbf{A}^\circ)$  by Lemma 5. Hence  $\text{CSP}(h(\mathbf{A}^\circ))$  is a so-called general subgroup problem and can be solved in polynomial time by a result of Feder and Vardi [8, Theorem 33]. Then  $\text{SysTerm}(\mathbf{A})$  is in P by Lemmas 4 and 6.

## 66:10 Satisfiability of Systems of Term Equations

Else if the core of  $\mathbf{A}$  is not abelian, then it has no idempotent Taylor polymorphism by Theorem 9. Hence  $\text{SysTerm}(\mathbf{A})$  is NP-complete by the hardness of CSP over structures without Taylor polymorphisms and Lemma 6. ◀

The proof of Corollary 10 does not require the full strength of the CSP-dichotomy by Bulatov and Zhuk but only that linear systems over modules are in P. Moreover, for an abelian algebra  $\mathbf{A}$  in a Taylor variety we can give a parametrization of all solutions of a system of term equations and determine their number by linear algebra in polynomial time. All of this applies in particular to algebras in congruence modular varieties.

**Proof of Theorem 2.** Let  $\mathbf{A}$  be a finite algebra in a congruence modular variety. Then the variety  $V$  generated by  $\mathbf{A}$  is locally finite and congruence modular. By [12, Theorem 8.5]  $V$  omits types 1 (i.e.,  $V$  is Taylor) and 5. In particular  $\mathbf{A}$  itself has no congruences  $\alpha \prec \beta$  of type 5. Hence the result is a special case of Corollary 10. ◀

### 5 Metaquestions about the complexity dichotomy

In the following we assume that algebras  $(A, f_1, \dots, f_m)$  are given by the graphs of their basic operations  $f_1, \dots, f_m$ . So, for  $n$  the maximum arity of  $f_1, \dots, f_m$ , this representation has size at least  $|A|^n$ .

We give a quasi-polynomial time algorithm to decide whether a given algebra in a Taylor variety has abelian core.

**Proof of Theorem 3.** Let  $\mathbf{A}$  be a finite algebra in a Taylor variety. First we claim that the core of  $\mathbf{A}$  is abelian iff there exists a homomorphism from the maximal abelian quotient  $\bar{\mathbf{A}} := \mathbf{A}/[1, 1]$  of  $\mathbf{A}$  to  $\mathbf{A}$ .

For the “only if”-direction, assume that the core  $h(\mathbf{A})$  for some endomorphism  $h$  of  $\mathbf{A}$  is abelian. By the Homomorphism Theorem  $h(\mathbf{A})$  is isomorphic to the quotient of  $\mathbf{A}$  by the kernel  $\ker h := \{(x, y) \in A^2 : h(x) = h(y)\}$  of  $h$ . In particular  $\mathbf{A}/\ker h$  is abelian as well. By the definition of the commutator,  $[1, 1]$  is the unique smallest congruence of  $\mathbf{A}$  such that  $\mathbf{A}/[1, 1]$  is abelian. Hence  $[1, 1] \leq \ker h$ . Let  $x/[1, 1]$  denote the class of  $x$  modulo  $[1, 1]$  in  $\bar{\mathbf{A}}$ . Then  $\bar{h}: \bar{\mathbf{A}} \rightarrow \mathbf{A}$ ,  $x/[1, 1] \mapsto h(x)$ , is a well-defined homomorphism.

Conversely, for the “if”-direction, assume we have a homomorphism  $\bar{h}: \bar{\mathbf{A}} \rightarrow \mathbf{A}$ . Then  $\bar{h}$  lifts to an endomorphism  $h: \mathbf{A} \rightarrow \mathbf{A}$ ,  $x \mapsto \bar{h}(x/[1, 1])$ . Clearly the images of  $h$  and  $\bar{h}$  are the same and the kernel  $\ker h$  contains  $[1, 1]$ . So by the Homomorphism Theorem  $h(\mathbf{A})$  is isomorphic to a quotient of  $\bar{\mathbf{A}}$ . Note that  $\bar{\mathbf{A}}$  is abelian in a Taylor variety and hence has a Mal’cev term operation  $d$  by [12, Theorem 9.6]. Hence  $\bar{\mathbf{A}}$  is polynomially equivalent to a module and all its quotients are abelian as well by [9] (We note in passing that outside of Taylor varieties, unfortunately quotients of abelian algebras may not be abelian again). In particular  $h(\mathbf{A})$  is abelian. While  $h(\mathbf{A})$  may not be the core of  $\mathbf{A}$ , all images  $g(\mathbf{A})$  for an endomorphism  $g$  of  $\mathbf{A}$  that are contained in  $h(\mathbf{A})$  are subalgebras of an abelian algebra, thus abelian themselves (This holds for arbitrary algebras by the definition of the commutator). Since the core of  $\mathbf{A}$  is isomorphic to a minimal such image  $g(\mathbf{A})$ , it is abelian.

Hence it suffices to check whether there exists a homomorphism from  $\bar{\mathbf{A}}$  to  $\mathbf{A}$ . Recall that the commutator  $[1, 1]$  can be enumerated in polynomial time by an algorithm due to Willard [7, Proposition 4.1]. Thus computation in  $\bar{\mathbf{A}}$  effectively reduces to computation in  $\mathbf{A}$  in the following steps.

Using the description of the structure of abelian algebras in congruence modular varieties from [9], we see that the subalgebra  $M_{\bar{\mathbf{A}}}(1, 1)$  of  $\bar{\mathbf{A}}^{2 \times 2}$  that is generated by all elements of the form

$$\begin{bmatrix} a & a \\ b & b \end{bmatrix}, \begin{bmatrix} c & d \\ c & d \end{bmatrix} \quad \text{for } a, b, c, d \in \bar{A}$$

has the universe

$$M_{\bar{\mathbf{A}}}(1, 1) = \left\{ \begin{bmatrix} y & x \\ z & d(x, y, z) \end{bmatrix} : x, y, z \in \bar{A} \right\}.$$

Hence the Mal'cev term operation  $d$  on  $\bar{\mathbf{A}}$  is unique and its graph can be computed by enumerating the elements in  $M_{\bar{\mathbf{A}}}(1, 1)$  by a straightforward closure algorithm in polynomial time. More specifically for any fixed element  $u_0 \in \bar{A}$ , the operations  $x + y := d(x, u_0, y)$  and  $-x := d(u_0, x, u_0)$  on  $\bar{A}$  yield an abelian group  $(\bar{A}, +, -, u_0)$  with zero element  $u_0$ . Further  $d(x, y, z) = x - y + z$  for all  $x, y, z \in \bar{A}$ . Now we can grow a generating set  $u_1, \dots, u_n$  of  $(\bar{A}, +)$  with  $n \leq \log_2 |\bar{A}|$  as follows. If  $u_0, \dots, u_i$  are fixed and the generated subgroup  $B_i := \langle u_0, \dots, u_i \rangle$  of  $(\bar{A}, +)$  is not all of  $\bar{A}$ , then pick some  $u_{i+1} \in \bar{A} \setminus B_i$ . Note that the index of  $B_i$  in  $B_{i+1}$  is at least 2 by Lagrange's Theorem. So the process stops with  $B_n = \bar{A}$  after  $n \leq \log_2 |\bar{A}|$  steps each of which requires only polynomial time in  $|A|$ . Finally we have obtained a generating set  $u_0, u_1, \dots, u_n$  for  $(\bar{A}, x - y + z)$  and in particular for  $\bar{\mathbf{A}}$  in polynomial time in  $|A|$ .

Clearly every homomorphism  $h: \bar{\mathbf{A}} \rightarrow \mathbf{A}$  is uniquely determined by its images on the generators  $u_0, \dots, u_n$ . For  $v_0, \dots, v_n \in A$  we can enumerate

$$\langle (u_0, v_0), \dots, (u_n, v_n) \rangle \leq \bar{\mathbf{A}} \times \mathbf{A}$$

in polynomial time to see whether the partial map with  $h(u_i) := v_i$  for  $i \in \{0, \dots, n\}$  extends to a homomorphism from  $\bar{\mathbf{A}}$  to  $\mathbf{A}$ . Checking the  $|A|^{n+1} \leq 2^{(\log |A|)^2 + \log |A|}$  potential images of  $u_0, \dots, u_n$  yields an algorithm that decides the existence of a homomorphism from  $\bar{\mathbf{A}}$  to  $\mathbf{A}$  in quasi-polynomial time in  $|A|$ .

If such a homomorphism exists, then the homomorphism with smallest image maps  $\bar{\mathbf{A}}$  to the abelian core of  $\mathbf{A}$  with Mal'cev term operation induced by  $d$  on  $\bar{\mathbf{A}}$ . Thus the universe of the core of  $\mathbf{A}$  can be obtained in quasi-polynomial time as well.  $\blacktriangleleft$

---

## References

- 1 L. Barto, A. Krokhin, and R. Willard. Polymorphisms, and how to use them. In Andrei A. Krokhin and Stanislav Zivný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/DFU.Vol17.15301.1.
- 2 C. Bergman. *Universal algebra*, volume 301 of *Pure and Applied Mathematics (Boca Raton)*. CRC Press, Boca Raton, FL, 2012. Fundamentals and selected topics.
- 3 P. Broniek. *Computational complexity of solving equation systems*. SpringerBriefs in Philosophy. Springer, Cham, 2015. doi:10.1007/978-3-319-21750-5.
- 4 A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
- 5 S. Burris and H. P. Sankappanavar. *A course in universal algebra*. Springer New York Heidelberg Berlin, 1981.

- 6 H. Chen and B. Larose. Asking the metaquestions in constraint tractability. *ACM Trans. Comput. Theory*, 9(3):Art. 11, 27, 2017. doi:10.1145/3134757.
- 7 W. DeMeo, R. Freese, and M. Valeriote. Polynomial-time tests for difference terms in idempotent varieties. *Internat. J. Algebra Comput.*, 29(6):927–949, 2019. doi:10.1142/S021819671950036X.
- 8 T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104 (electronic), 1999. doi:10.1137/S0097539794266766.
- 9 R. Freese and R. N. McKenzie. *Commutator Theory for Congruence Modular Varieties*, volume 125 of *London Math. Soc. Lecture Note Ser.* Cambridge University Press, 1987. Available from <http://math.hawaii.edu/~ralph/Commutator/comm.pdf>.
- 10 R. Freese and M. A. Valeriote. On the complexity of some Maltsev conditions. *Internat. J. Algebra Comput.*, 19(1):41–77, 2009. doi:10.1142/S0218196709004956.
- 11 M. Goldmann and A. Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178(1):253–262, 2002. doi:10.1006/inco.2002.3173.
- 12 D. Hobby and R. McKenzie. *The structure of finite algebras*, volume 76 of *Contemporary mathematics*. American Mathematical Society, 1988.
- 13 K. Kearnes, P. Marković, and R. McKenzie. Optimal strong Mal’cev conditions for omitting type 1 in locally finite varieties. *Algebra Universalis*, 72(1):91–100, 2014. doi:10.1007/s00012-014-0289-9.
- 14 O. Klíma, P. Tesson, and D. Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory Comput. Syst.*, 40(3):263–297, 2007. doi:10.1007/s00224-005-1279-2.
- 15 B. Larose and L. Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Internat. J. Algebra Comput.*, 16(3):563–581, 2006. doi:10.1142/S0218196706003116.
- 16 R. N. McKenzie, G. F. McNulty, and W. F. Taylor. *Algebras, lattices, varieties, Volume I*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, 1987.
- 17 M. Siggers. A strong Mal’cev condition for locally finite varieties omitting the unary type. *Algebra Universalis*, 64(1-2):15–20, 2010. doi:10.1007/s00012-010-0082-3.
- 18 D. Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

# Parallel Enumeration of Parse Trees

Margarita Mikhelson ✉

Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

Alexander Okhotin ✉ 

Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

---

## Abstract

A parallel algorithm for enumerating parse trees of a given string according to a fixed context-free grammar is defined. The algorithm computes the number of parse trees of an input string; more generally, it applies to computing the weight of a string in a weighted grammar. The algorithm is first implemented on an arithmetic circuit of depth  $O((\log n)^2)$  with  $O(n^6)$  elements. Then, it is improved using fast matrix multiplication to use only  $O(n^{5.38})$  elements, while preserving depth  $O((\log n)^2)$ .

**2012 ACM Subject Classification** Theory of computation → Grammars and context-free languages; Theory of computation → Parallel algorithms

**Keywords and phrases** Context-free grammars, weighted grammars, parsing, parallel algorithms, matrix multiplication

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.67

**Funding** This work was supported by the Russian Science Foundation, project 23-11-00133.

## 1 Introduction

The classical cubic-time Cocke–Kasami–Younger parsing algorithm can be adapted to solving such problems as counting the number of parse trees of an input string, or computing the probability of an input string in a probabilistic grammar – and, more generally, computing the weight of a string in a weighted grammar. This is possible, because the algorithm considers the substrings in a certain predictable order, and investigates each partition of each substring exactly once, and hence never enumerates the same parse tree twice.

Another classical parsing algorithm discovered by Valiant [9] computes the same values in the time of matrix multiplication  $O(n^\omega)$ , where  $\omega < 3$ . In spite of quite a nontrivial order of processing the substrings and their partitions, it maintains the same property: every partition of every substring is considered only once, there is no double counting. Hence, the algorithm can be applied to computing weights in any ring [2], see Okhotin [6] for an elementary presentation.

These are the sequential parsing algorithms. Yet another classical parsing method is a parallel algorithm discovered independently by Brent and Goldschlager [3] and by Rytter [7]. Their algorithm, if implemented on a parallel architecture with concurrent reads and exclusive writes, works in time  $O((\log n)^2)$  and uses  $O(n^6)$  processors. If implemented on a uniform family of Boolean circuits (the model of parallel computation assumed in this paper), the algorithm by Brent, Goldschlager and Rytter uses circuits of depth  $O((\log n)^2)$  with  $O(n^6 \log n)$  elements, that is, NC<sup>2</sup>-circuits.

What is important to know about the Brent–Goldschlager–Rytter parallel parsing algorithm is that it *considers the same trees multiple times*, and this happens in different branches of parallel computation: each tree is obtained in different ways by combining different subtrees. An example of this is given in Figure 1: here the input string is  $a_1a_2a_3a_4$ , and the algorithm obtains its single parse tree in two ways. One way is to produce a subtree of  $a_1a_2$  with root  $A$ , another tree with root  $S$  with a “hole” instead of  $A$  and with the



© Margarita Mikhelson and Alexander Okhotin;  
licensed under Creative Commons License CC-BY 4.0

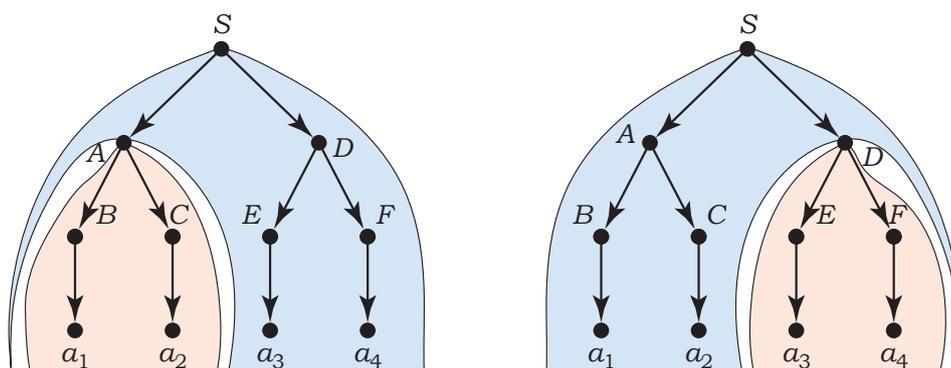
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 67; pp. 67:1–67:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Multiple partitions of a tree into a “tree with a hole” and a subtree.

leaves  $a_3a_4$  to the right of the hole, and then to fill this hole with the former subtree, see Figure 1(left). The other way is similar, using a subtree of  $a_3a_4$  with root  $D$  to fill a hole in another “tree with a hole”. The algorithm will construct these subtrees in different branches of parallel computation, each time deducing the existence of a parse tree of  $a_1a_2a_3a_4$ . If the algorithm were modified to count the number of trees, then the tree in the figure will be counted twice, as the algorithm has no means to check whether these trees were the same or not. Hence, the algorithm can only determine the existence of a tree, but cannot compute a reliable count of the number of trees, as well as cannot compute the weight of the input string in most semirings.

In a recent paper by Bakinova et al. [1], a simple variant of the Brent–Goldschlager–Rytter algorithm for computing *the number of trees modulo two* was presented: the algorithm uses a Boolean circuit of the same depth  $O((\log n)^2)$ , whereas the number of elements is  $O(n^7)$ . This algorithm considers each tree only once, and therefore it can be adapted to compute the precise number of trees, if the Boolean circuit is replaced with an arithmetic circuit. In the same way, one can compute the weight in any semiring. This raises a natural question: could the number of elements in  $(\log n)^2$ -depth circuits solving this problem be reduced to  $O(n^6)$  or below?

This paper presents a new parallel algorithm for enumerating all parse trees of an input string, which maintains the same circuit depth  $O(\log^2 n)$ , and is more efficient in terms of the number of elements. First, in Section 3, the problem is solved using a circuit with  $O(n^6)$  elements. The algorithm is formulated as computing the number of trees, and then it is adapted to compute the weight of an input string in any semiring.

Next, in Section 4, this algorithm is improved by using fast matrix multiplication. The algorithm elaborates the algorithm in Section 3 in the same way as Valiant’s [9] algorithm elaborates the Cocke–Kasami–Younger algorithm: it computes the same values as the original algorithm, but rearranges the order of computation so that the arithmetical operations are combined into matrix multiplications. As compared to Valiant’s [9] algorithm, the proposed algorithm has to handle trees with holes, which require a more difficult rearrangement of the computation.

By abuse of notation, let  $\omega \geq 2$  be a number, such that for every  $n$  and for every ring there is a circuit of depth  $O(\log n)$  with  $O(n^\omega)$  elements computing operations in the ring, which multiplies any two  $n \times n$  matrices. Strassen’s algorithm [8] can be implemented on such a circuit. Also it is known that the Coppersmith–Winograd [4] algorithm can be implemented on a logarithmic-depth circuit as well – see, e.g., Gazit and Miller [5]. Then

the new algorithm for enumerating parse trees maintains circuit depth  $O(\log^2 n)$ , whereas the number of elements in it is reduced to  $O(n^{\omega+3})$ , that is, does not exceed  $O(n^{5.38})$ . The algorithm is applicable to computing the weight of a string in any ring.

## 2 Definitions

► **Definition 1.** *A grammar in the Chomsky normal form is a quadruple  $G = (\Sigma, N, R, S)$ , where  $\Sigma$  is a finite alphabet;  $N$  is a finite set of nonterminal symbols;  $R$  is a finite set of grammar rules, each of the form  $A \rightarrow BC$ , with  $A, B, C \in N$ , or  $A \rightarrow a$ , with  $A \in N$  and  $a \in \Sigma$ ;  $S \in N$  is the initial symbol.*

► **Definition 2.** *Let  $G = (\Sigma, N, R, S)$  be a grammar in the Chomsky normal form. A parse tree in  $G$  is a tree, in which the root is labelled with  $S$ , every internal vertex is labelled with any nonterminal symbol  $A \in N$ , and every leaf is labelled with a symbol  $a \in \Sigma$ . Every vertex  $A \in N$  may either have two sons labelled  $B$  and  $C$ , for some rule  $A \rightarrow BC$ , or one son labelled  $a$ , if there is a rule  $A \rightarrow a$ . Vertices with two sons have their sons ordered, which induces an order on the leaves. If  $w \in \Sigma^+$  is the string formed by the leaves of a tree, this is a parse tree of  $w$ .*

*The language defined by the grammar, denoted by  $L(G)$  is the set of all strings  $w \in \Sigma^+$  that have at least one parse tree.*

Under this definition, it is easy to generalize grammars to weighted grammars over a semiring.

► **Definition 3.** *Let  $G = (\Sigma, N, R, S)$  be a grammar in the Chomsky normal form, let  $\mathcal{S}$  be a semiring, and let  $\Phi: R \rightarrow \mathcal{S}$  be a function assigning weights in  $\mathcal{S}$  to rules. The triple  $(G, \mathcal{S}, \Phi)$  is called a weighted grammar.*

*Then, the weight of a parse tree is the product of weights of all rules used in the tree. The weight of a string  $w \in \Sigma^*$  is the sum of weights of all parse trees of  $w$ . The grammar defines a mapping from  $\Sigma^*$  to  $\mathcal{S}$ .*

If  $\mathcal{S}$  is the Boolean semiring and all rules have weight 1, then a weighted grammar become an ordinary grammar, which defines the language  $\{w \mid w \text{ has weight } 1\}$ ; and if  $\mathcal{S}$  is a two-element field, the weighted grammar becomes a GF(2)-grammar. If  $\mathcal{S}$  is the set of non-negative integers with addition and multiplication, and all rules have weight 1, then the weighted grammar defines the number of parse trees in  $G$ . If  $\mathcal{S}$  is the set of probabilities in  $[0, 1]$  with addition and multiplication, this is a probabilistic grammar.

This paper uses arithmetic circuits as a model of parallel computation.

► **Definition 4.** *An arithmetic circuit over natural numbers  $\mathbb{N} = \{0, 1, 2, \dots\}$  is a directed acyclic graph, in which all nodes with no incoming arcs are called the inputs and are labelled with distinct input variables, and each of the rest of the nodes, called elements, has two incoming arcs and is labelled either with addition, or with multiplication. Input variables are natural numbers, each element computes a function of the input variables. The circuit is said to compute the function computed in a designated output element.*

*An arithmetic circuit over integers  $\mathbb{Z}$  additionally may use the subtraction operation. Arithmetic circuits over a semiring  $\mathcal{S}$  and over a ring  $\mathcal{R}$  generalize these circuits by computing values in the corresponding algebraic structures.*

*The depth of a circuit is the length of the longest path.*

A parsing algorithm is implemented on a circuit, which is constructed for a fixed grammar  $G = (\Sigma, N, R, S)$  and a fixed length of input strings  $n$ . The circuit has  $|\Sigma| \cdot n$  inputs, each indicating whether there is a particular symbol in a particular position. The output element gives the number of parse trees of the given string.

Since a different circuit is needed for each input length, the following standard uniformity condition is assumed.

► **Definition 5.** A family of arithmetic circuits is uniform, if there exists a deterministic logarithmic-space Turing machine that, for every input length  $n$  given in unary notation, produces the circuit for input strings of length  $n$ .

### 3 A circuit with $O(n^6)$ elements

► **Theorem 6.** For every grammar  $G = (\Sigma, N, R, S)$  in the Chomsky normal form there is a uniform family of arithmetic circuits, for each length of input strings  $n \geq 1$ , which computes the number of parse trees of an input string of length  $n$ , has at most  $|N|^2 \cdot |R| \cdot n^6$  elements and is of depth at most  $21 \log_2^2 n + 7 \log_2 n \log_2 |R|$ .

**Proof.** Denote  $w = a_1 \dots a_n$ . The circuit consists of the following elements.

- An element  $A(i, j)$ , for all  $A \in N$  and  $i, j$ , with  $0 \leq i < j \leq n$ , computes the number of parse trees from  $A$  for the string  $a_{i+1} \dots a_j$ ; these trees are schematically presented as in Figure 2(left).

An example of such a tree for  $A(0, 2)$  can be seen in Figure 1(left).

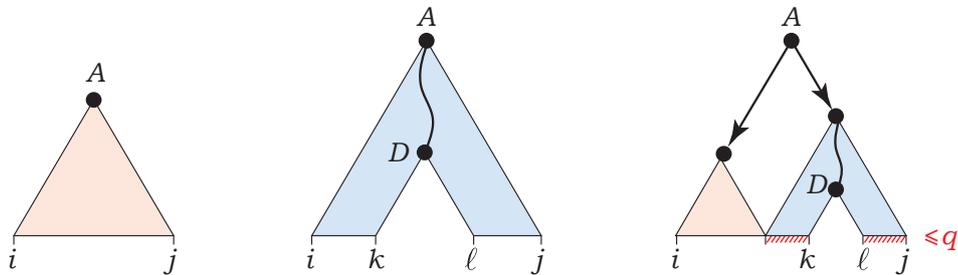
- An element  $\frac{A}{D}(i, k, \ell, j)$ , for all  $A, D \in N$  and  $0 \leq i \leq k < \ell \leq j \leq n$ , computes the number of parse trees for the string  $a_{i+1} \dots a_j$  from  $A$ , with a hole instead of a subtree  $a_{k+1} \dots a_\ell$  from  $D$ , depicted as in Figure 2(middle).

Such a “tree with a hole” for  $\frac{S}{A}(0, 0, 2, 4)$  can be seen in Figure 1(left).

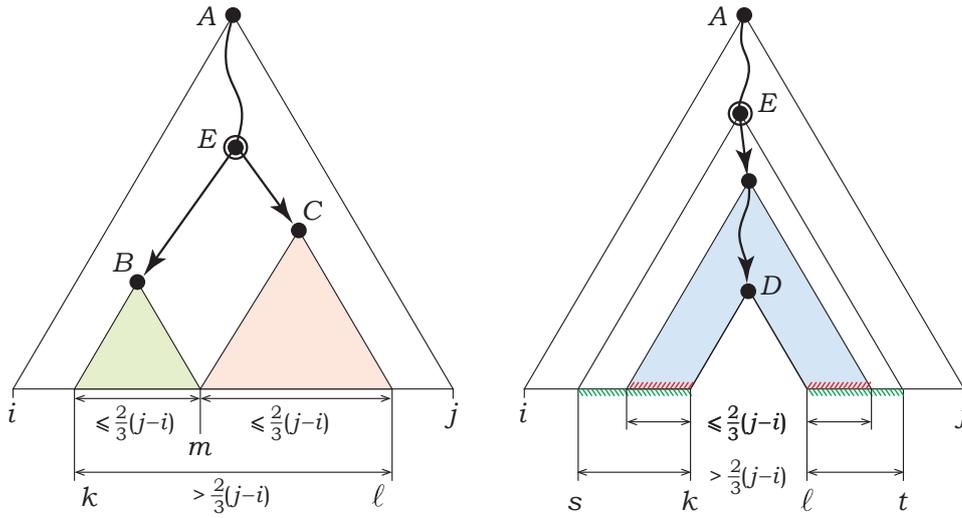
- An element  $\frac{A}{D}|_q(i, k, \ell, j)$ , for all  $A, D \in N$  and  $0 \leq i \leq k < \ell \leq j \leq n$ , with  $(k - i) + (j - \ell) > 0$  and  $q \geq 0$ , computes the number of parse trees for the string  $a_{i+1} \dots a_j$  from  $A$ , with a hole instead of a subtree  $a_{k+1} \dots a_\ell$  from  $D$ , such that at the first branching the subtree with the hole contains at most  $q$  leaves, as shown in Figure 2(right).

The tree in the earlier Figure 1(left) is too small to illustrate this, because after the first branching the subtree with the hole is empty (that is, contains 0 leaves).

- Some extra elements are used to compute the values of the named elements listed above: e.g., each element  $A(i, j)$  is computed by a circuit that consists of  $O(n^3)$  extra elements, etc.



■ **Figure 2** (left)  $A(i, j)$ ; (middle)  $\frac{A}{D}(i, k, \ell, j)$ ; (right)  $\frac{A}{D}|_q(i, k, \ell, j)$ .



■ **Figure 3** (left) Intermediate vertex  $E$  in a tree  $A(i, j)$ ; (right) Intermediate vertex  $E$  on the path to the hole in a tree  $\frac{A}{D}(i, k, \ell, j)$ .

Denote the inputs of the circuit by  $In(a, i)$ , for all  $a \in \Sigma$  and  $1 \leq i \leq n$ , where  $In(a, i) = 1$  if the  $i$ -th symbol of an input string is equal to  $a$  and  $In(a, i) = 0$  otherwise. Elements of the circuit corresponding to one-symbol strings are initialized directly from the inputs as  $A(i - 1, i) = \sum_{A \rightarrow a \in R} In(a, i)$ , for all  $A \in N$  and  $1 \leq i \leq n$ . Also, trees with a hole containing no leaves are defined for every pair of positions:  $\frac{A}{D}(i, i, j, j) = 1$ , for  $A, D \in N$  and  $0 \leq i < j \leq n$ .

The desired number of parse trees of  $w$  is computed by the element  $S(0, n)$ .

Consider an arbitrary parse tree  $\tau$ , corresponding to  $A(i, j)$  which has  $j - i$  leaves. Denote by  $E$  the deepest vertex having more than  $\frac{2}{3}(j - i)$  leaves in its subtree. Such a vertex exists, moreover it is unique. Denote by  $B$  and  $C$  the children of  $E$ . Suppose that the subtree of  $B$  corresponds to leaves  $a_{k+1} \dots a_m$ , and the subtree of  $C$  corresponds to leaves  $a_{m+1} \dots a_\ell$ , as illustrated in Figure 3(left). Taking into account that  $E$  is the deepest vertex with more than  $\frac{2}{3}(j - i)$  leaves, both  $B$  and  $C$  may only have at most  $\frac{2}{3}(j - i)$  leaves each, in other words,  $\ell - k > \frac{2}{3}(j - i)$  and  $\max(m - k, \ell - m) \leq \frac{2}{3}(j - i)$ .

Hence, the number of parse trees for  $A(i, j)$  can be calculated as

$$A(i, j) = \sum_{\substack{E \rightarrow BC \in R \\ k, \ell, m: i \leq k < m < \ell \leq j \\ \ell - k > \frac{2}{3}(j - i) \\ \max(m - k, \ell - m) \leq \frac{2}{3}(j - i)}} \frac{A}{E}(i, k, \ell, j) \cdot B(k, m) \cdot C(m, \ell)$$

Every parse tree  $\tau$  has been taken into account, because there is such a vertex  $E$  in every tree. Furthermore, the vertex  $E$  is unique for the given parse tree  $\tau$ , therefore every  $\tau$  is taken into account only once.

The above formula for  $A(i, j)$  is implemented on a circuit as follows. For each triple  $(k, \ell, m)$ , one extra element is used to multiply  $\frac{A}{E}(i, k, \ell, j)$  by  $B(k, m)$ , and another extra element multiplies the result by  $C(m, \ell)$ . It remains to sum up the resulting products: as there are  $O(n^3)$  of them, up to  $O(n^3)$  extra elements are needed, and they are organized into a tree of logarithmic depth.

Similar arguments allow us to deduce a formula for  $\frac{A}{D}(i, k, \ell, j)$  and to implement it on a circuit. Denote by  $E$  the deepest vertex on the path from  $A$  to  $D$  with more than  $\frac{2}{3}(k - i + j - \ell)$  leaves in its subtree, and denote these leaves by  $a_{s+1} \dots a_k$  and  $s_{\ell+1} \dots s_t$ , see Figure 3(right). Then

$$\frac{A}{D}(i, k, \ell, j) = \sum_{\substack{E \in N \\ s: i \leq s \leq k \\ t: \ell \leq t \leq j \\ k-s+t-\ell > \frac{2}{3}(k-i+j-\ell)}} \frac{A}{E}(i, s, t, j) \cdot \frac{E}{D} \Big|_{\frac{2}{3}(k-i+j-\ell)}(s, k, \ell, t)$$

Indeed, the intermediate positions  $s$  and  $t$  are chosen so that the subtree  $E$  has more than  $\frac{2}{3}(k - i + j - \ell)$  leaves. On the other hand, the definition of  $\frac{E}{D} \Big|_{\frac{2}{3}(k-i+j-\ell)}$  guarantees that no other vertex on the path from  $E$  to  $D$  has more than  $\frac{2}{3}(k - i + j - \ell)$  leaves. Hence,  $E$  is indeed the deepest vertex on the path from  $A$  to  $D$  with more than  $\frac{2}{3}(k - i + j - \ell)$  leaves. Since such a vertex is unique for each tree, every parse tree for  $\frac{A}{D}(i, k, \ell, j)$  is counted only once.

A circuit computing  $\frac{A}{D}(i, k, \ell, j)$  by the above formula uses  $O(n^2)$  extra elements.

In order to compute  $\frac{E}{D} \Big|_q(s, k, \ell, t)$ , let us consider the rule used at the root  $E$  of a tree with a hole  $D$ . Let  $E \rightarrow BC$  be this rule, and let  $m$  be the position in the input such that leaves up to  $a_m$  belong to subtree  $B$ , and leaves beginning with  $a_{m+1}$  belong to subtree  $C$ . Then  $\frac{E}{D} \Big|_q(s, k, \ell, t)$  can be computed as follows:

$$\frac{E}{D} \Big|_q(s, k, \ell, t) = \sum_{\substack{E \rightarrow BC \in R \\ m: s < m \leq k \\ k-m+t-\ell \leq q}} B(s, m) \cdot \frac{C}{D}(m, k, \ell, t) + \sum_{\substack{E \rightarrow BC \in R \\ m: \ell \leq m < t \\ k-s+m-\ell \leq q}} \frac{B}{D}(s, k, \ell, m) \cdot C(m, t)$$

A circuit computing an element  $\frac{E}{D} \Big|_q(s, k, \ell, t)$  by this formula uses  $O(n)$  extra elements.

The entire circuit consists of all the above elements and computes the desired values.

**Number of elements in the circuit.** There are  $O(n^2)$  elements  $A(i, j)$  and computing each of them takes  $O(n^3)$  extra elements. Similarly, each of the  $O(n^4)$  elements  $\frac{A}{D}(i, k, \ell, j)$  takes  $O(n^2)$  extra elements to compute, whereas the computation of each of the  $O(n^5)$  elements  $\frac{E}{D}(s, k, \ell, t) \Big|_q$  takes  $O(n)$  extra elements. This accounts for  $O(n^6)$  elements in the entire circuit; the calculation of the dependence on the grammar is omitted due to space constraints.

**Depth of the circuit.** The elements  $A(i, j)$  directly depend on elements, which have at most  $\frac{2}{3}(j - i)$  leaves in their subtrees.

For each element  $\frac{A}{D}(i, k, \ell, j)$ , let  $\alpha = k - i + j - \ell$  be the number of leaves in its subtree. Then it directly depends on elements  $\frac{E}{D}(\cdot, \cdot, \cdot, \cdot)$  with at most  $\frac{1}{3}\alpha$  leaves, and elements  $\frac{E}{D} \Big|_q(\cdot, \cdot, \cdot, \cdot)$  with at most  $\alpha$  leaves, and with  $q \leq \frac{2}{3}\alpha$ . The latter element depends on  $B(\cdot, \cdot)$  with at most  $\alpha$  leaves, and on  $\frac{C}{D}(\cdot, \cdot, \cdot, \cdot)$  with at most  $\frac{2}{3}\alpha$ . Overall,  $\frac{A}{D}$  depends on elements with at most  $\frac{2}{3}\alpha$  leaves in three steps.

Each element  $\frac{E}{D} \Big|_q(s, k, \ell, t)$  depends on elements  $\frac{A}{D}$  and  $B$  with fewer than  $\alpha$  leaves.

Thus, for an element of every type, whether it is  $A(\cdot, \cdot)$ ,  $\frac{E}{D} \Big|_q(\cdot, \cdot, \cdot, \cdot)$  or  $\frac{A}{D}(\cdot, \cdot, \cdot, \cdot)$ , after a quadruple application of the rules, the number of leaves is reduced at least by a factor of  $\frac{3}{2}$ .

At each level of these dependencies, a sum of multiple values is computed using a binary tree of logarithmic depth in the number of arguments, whence overall depth  $O(\log^2 n)$ . ◀

The above circuit actually performs operations in the ring of natural numbers, adding constant 1 for each parse tree accounted for. This can be generalized to compute the value of a string in a weighted grammar over every semiring.

► **Theorem 7.** *For every weighted grammar  $G = (\Sigma, N, R, S)$  in the Chomsky normal form, with weights in a semiring  $\mathcal{S}$ , there is a uniform family of arithmetic circuits computing elements over  $\mathcal{S}$ , for each length of input strings  $n \geq 1$ , which computes the value of an input string of length  $n$  in the semiring, has  $O(n^6)$  elements and is of depth  $O(\log^2 n)$ .*

## 4 A smaller circuit using fast matrix multiplication

In the circuit constructed in Theorem 6, all elements  $A(\cdot, \cdot)$  are computed using  $\Theta(n^5)$  operations in total, and computation of elements of both types  $\frac{A}{D}$  and  $\frac{A}{D}|_q$  takes  $\Theta(n^6)$  operations. Hence, in order to reduce the total number of elements in the circuit, computation of elements  $\frac{A}{D}$  and  $\frac{A}{D}|_q$  should be optimized. It turns out that such an optimization could be done via fast matrix multiplication.

### 4.1 Matrix multiplication for $\frac{A}{D}$

Let us show how fast matrix multiplication can be used to compute the same values  $\frac{A}{D}(i, k, \ell, j)$  using fewer elements.

In the following, matrices with rows and columns indexed by *substrings*, that is, pairs of two positions in the string, will be considered. Denote  $I = (i, j)$  and  $|I| = j - i$ ; similarly, let  $K = (k, \ell)$ ,  $|K| = \ell - k$ ,  $S = (s, t)$ ,  $|S| = t - s$ . For every pair of non-terminal symbols  $A, D$  let us define a matrix of size  $\frac{n(n+1)}{2} \times \frac{n(n+1)}{2}$ , consisting of elements  $\frac{A}{D}[I][K] = \frac{A}{D}(i, k, \ell, j)$  with  $i \leq k < \ell \leq j$  and  $\frac{A}{D}[I][K] = 0$  otherwise. Then the formula for computing the value  $\frac{A}{D}(i, k, \ell, j)$  can be rewritten in the new notation as follows.

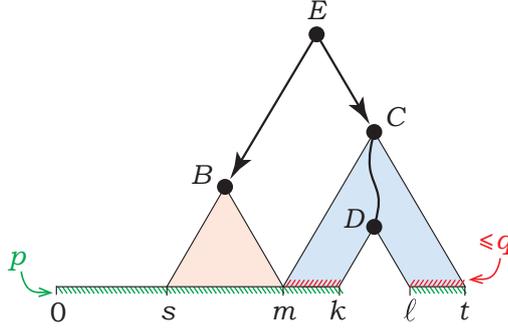
$$\frac{A}{D}[I][K] = \sum_{\substack{E \in N \\ S: K \subset S \subset I \\ |S| - |K| > \frac{2}{3}(|I| - |K|)}} \frac{A}{E}[I][S] \cdot \frac{E}{D}|_{\frac{2}{3}(|I| - |K|)}[S][K]$$

In this notation, the formula resembles a matrix product, and suggests that some elements of a matrix  $\frac{A}{D}[I][K]$ , with rows indexed by  $I$  and columns indexed by  $K$ , could be computed as a product of some submatrices of  $\frac{A}{E}$  and of  $\frac{E}{D}|_{\frac{2}{3}(|I| - |K|)}$  (as long as  $|I| - |K|$  is fixed). However, an algorithm cannot compute the product of these matrices in one step, since the elements corresponding to the longer substrings indirectly depend on the elements corresponding to the shorter substrings. For this reason, those matrices shall be split into several submatrices, which shall be multiplied separately at different stages of the computation.

The matrix  $\frac{A}{D}[I][K]$  is split into submatrices with fixed  $|I|$  and  $|K|$ . The submatrix with  $|I| = \alpha$  and  $|K| = \beta$ , is denoted by  $\left\{ \frac{A}{D}[I][K] \right\}_{|I|=\alpha, |K|=\beta}$ , and is of size  $(n - \alpha + 1) \times (n - \beta + 1)$ . This submatrix is computed as the following sum of matrix products.

$$\left\{ \frac{A}{D}[I][K] \right\}_{\substack{|I|=\alpha \\ |K|=\beta}} = \sum_{\substack{E \in N \\ \gamma: \frac{2}{3}\alpha + \frac{1}{3}\beta < \gamma < \alpha}} \left\{ \frac{A}{E}[I][S] \right\}_{\substack{|I|=\alpha \\ |S|=\gamma}} \cdot \left\{ \frac{E}{D}|_{\frac{2}{3}(\alpha - \beta)}[S][K] \right\}_{\substack{|S|=\gamma \\ |K|=\beta}}$$

(the condition  $\frac{2}{3}\alpha + \frac{1}{3}\beta < \gamma < \alpha$  is obtained from the inequalities  $|S| - |K| > \frac{2}{3}(|I| - |K|)$  and  $|S| < |I|$ )



■ **Figure 4** The computation of  $\frac{E}{D}\Big|_q(s, k, \ell, t)$  in the case of branching to the left.

The submatrices being multiplied include some elements  $\frac{A}{E}[I][S]$  with  $S \subsetneq I$ , and  $\frac{E}{D}\Big|_{\frac{2}{3}(\alpha-\beta)}[S][K]$  with  $K \subsetneq S$ , which are not in the formula for  $\frac{A}{D}[I][K]$ ; however, since these elements are zero by the definition of the matrices, these unintended terms in the sums have no effect on the result.

Also note that the computation of the element  $\frac{A}{D}[I][K]$  with  $\alpha - \beta$  leaves, requires only the elements  $\frac{A}{E}[\cdot][\cdot]$  with at most  $\frac{1}{3}(\alpha - \beta)$  leaves, and the elements  $\frac{E}{D}\Big|_q[\cdot][\cdot]$  with at most  $\alpha - \beta$  leaves. The overall order of computation will be set accordingly, so that by the time the product is computed, its arguments will have already been computed.

## 4.2 Matrix multiplication for elements $\frac{E}{D}\Big|_q$

Let us now consider the elements  $\frac{E}{D}\Big|_q(s, k, \ell, t)$ . As it was mentioned above, each of these elements can be calculated as

$$\frac{E}{D}\Big|_q(s, k, \ell, t) = \sum_{\substack{E \rightarrow BC \in R \\ m: s < m \leq k \\ k - m + t - \ell \leq q}} B(s, m) \cdot \frac{C}{D}(m, k, \ell, t) + \sum_{\substack{E \rightarrow BC \in R \\ m: \ell \leq m < t \\ k - s + m - \ell \leq q}} \frac{B}{D}(s, k, \ell, m) \cdot C(m, t)$$

Let us show how fast matrix multiplication can be used in order to compute the first sum. The second sum is computed similarly.

For each element  $\frac{E}{D}\Big|_q(s, k, \ell, t)$ , define a value  $p$ , which equals  $p = k + t - \ell$ , as it is shown in Figure 4. The proposed order of computations is designed in a such way that elements  $\frac{E}{D}\Big|_q$  with the same  $p$  are calculated together.

Let us fix  $p > 0$  and consider computations of  $\frac{E}{D}\Big|_q(s, k, \ell, t)$  with  $t - \ell + k = p$ . Denote a matrix  $U^{p,B}$  of size  $p \times p$ , such that for all  $s, m = 0, 1, \dots, p-1$ ,  $U^{p,B}[s][m] = B(s, m)$  if  $s < m$ , and  $U[s][m] = 0$  otherwise. In other words,  $U$  is a top-left corner of the matrix  $B$ . Also denote a matrix  $W^{p,q,C,D}$  of size  $p \times n^2$ , where the first coordinate is  $m = 0, 1, \dots, p-1$ , and the second is a triple of  $k, \ell, t$ , such that  $k + t - \ell = p$ , and  $W^{p,q,C,D}[m][k, \ell, t] = \frac{C}{D}(m, k, \ell, t)$  if  $m \leq k \leq \ell \leq t$  and  $m \geq p - q$ , and  $W^{p,q,C,D}[m][k, \ell, t] = 0$  otherwise.

It is claimed that the product of  $U^{p,B}$  and  $W^{p,q,C,D}$  contains the exact values needed for the computation of  $\frac{E}{D}\Big|_q$ .

$$(U^{p,B} \cdot W^{p,q,C,D})[s][k, \ell, t] = \sum_{m=0}^{p-1} U^{p,B}[s][m] \cdot W^{p,q,C,D}[m][k, \ell, t]$$

This sum has  $U^{p,B}[s][m] = 0$  for  $m \leq s$ , and  $W^{p,q,C,D}[m][k, \ell, t] = 0$  for  $m \geq k$  and for  $m \leq p - q$ . Since  $p = k + t - \ell \geq k$ , the upper limit  $m < p$  becomes redundant. Therefore, the range of the sum can be reduced as follows.

$$\begin{aligned} (U^{p,B} \cdot W^{p,q,C,D})[s][k, \ell, t] &= \sum_{m=0}^{p-1} U^{p,B}[s][m] \cdot W^{p,q,C,D}[m][k, \ell, t] = \\ &= \sum_{\substack{s < m \leq k \\ p-q \leq m}} B(s, m) \cdot \frac{C}{D}(m, k, \ell, t) \end{aligned}$$

Hence, in order to compute all elements  $\frac{E}{D}|_q(s, k, \ell, t)$  with  $t - \ell + k = p$ , it is needed to iterate over all pairs of non-terminals  $B, C$ , such that  $E \rightarrow BC \in R$ , construct the corresponding matrices  $U^{p,B}$  and  $W^{p,q,C,D}$ , and sum up their products. However, elements in each of these matrices indirectly depend on each other, so it is again impossible to build and multiply these matrices in one step. For this reason, matrices  $U^{p,B}$  and  $W^{p,q,C,D}$  are never built wholly, and their products are gradually calculated by multiplying their submatrices of various size. The higher the value of  $p$ , the later the entire product  $U^{p,B} \cdot W^{p,q,C,D}$  will be obtained. Let us also mention that both matrices  $U^{p,B}$  and  $W^{p,q,C,D}$ , consist of elements  $B, \frac{C}{D}$  and  $\frac{E}{D}|_q$  or zeros, and so does their product  $U^{p,B} \cdot W^{p,q,C,D}$ . Hence, there is no need to actually store these matrices – it is only a way to represent computations performed over the named elements.

► **Remark 8.** From the definition of  $U^{p,B}$  it follows that an arbitrary element  $U^{p,B}[s][m]$  has  $m - s$  leaves in its subtree. In other words, the elements with  $\alpha$  leaves in their subtrees lay on the  $\alpha$ -th diagonal, in the numeration where the main diagonal has number 0.

► **Remark 9.** Similarly, every element  $W^{p,q,C,D}[m][k, \ell, t]$  has  $\min(k - m + t - \ell, q) = \min(p - m, q)$  leaves in its subtree. In other words, the elements in the  $\alpha$ -th row from the bottom of  $W^{p,q,C,D}$  have  $\min(\alpha, q)$  leaves in their subtrees.

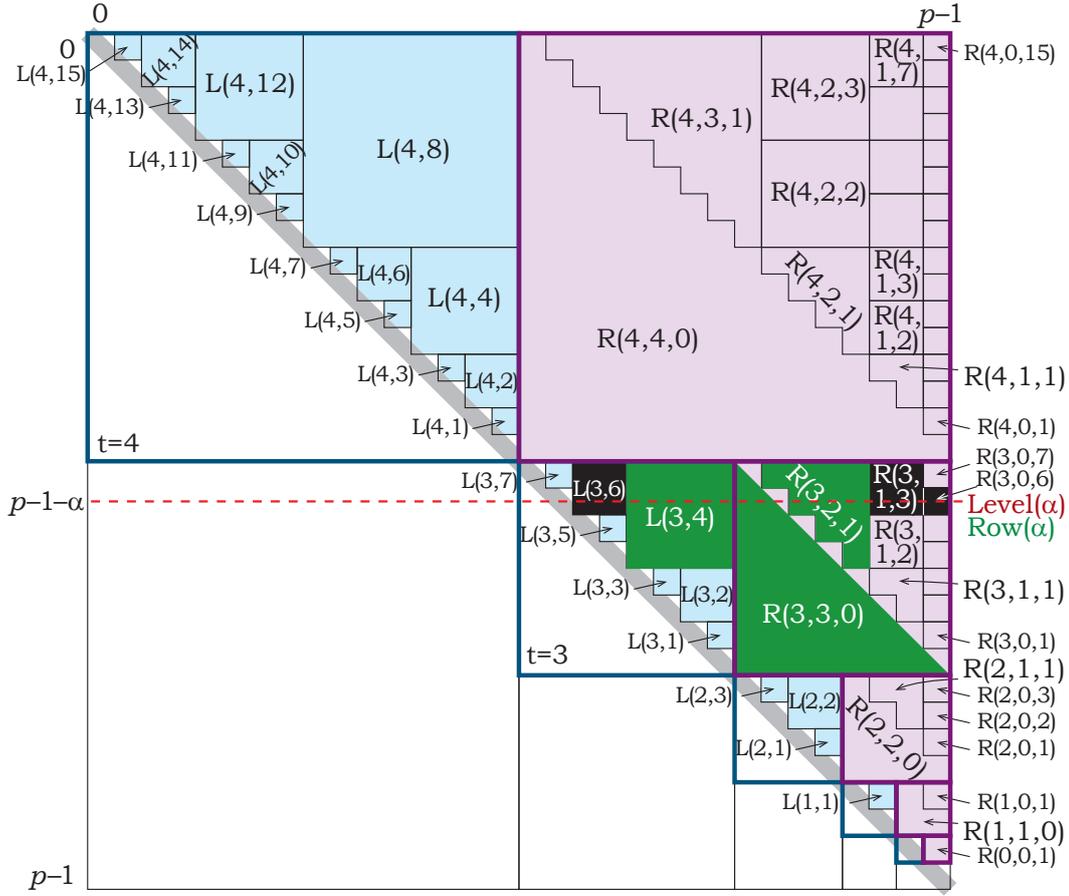
For a fixed value of  $p$ , the matrix  $U^{p,B}$  is split into disjoint submatrices, which will be multiplied by the algorithm. These matrices are denoted by  $L(t, j)$  and  $R(t, \ell, j)$ , where the number  $t$  refers to a large square of size  $2^t \times 2^t$ , in which the corresponding submatrix is contained. The L-matrices are in the large squares centered at the main diagonal, whereas the R-matrices are in the large squares to the right of the main diagonal. This decomposition is illustrated in Figure 5.

It is important to note that even though the matrices  $U^{p,B}$  for different  $p$  are corners of the same matrix  $B$ , the matrix  $U^{p,B}$  is split anew for each  $p$ , and there is no relation between the decompositions for different  $p$ .

► **Definition 10.** For all  $t = 0, \dots, \log_2 p - 1$  and  $j = 1, \dots, 2^t - 1$ , let  $n_j$  be the maximal power of two that divides  $j$ . Denote by  $L(t, j)$  the submatrix of matrix  $U^{p,B}$  of size  $n_j \times n_j$  with left bottom corner at  $(p - 2^t - j - 1, p - 2^t - j)$ .

$$L(t, j) = U^{p,B}[p - 2^t - j - n_j, \dots, p - 2^t - j - 1][p - 2^t - j, \dots, p - 2^t - j + n_j - 1]$$

Each  $L(t, j)$  is in the  $t$ -th large square, its left corner is next to the main diagonal, and  $j$  is the number of the row within the large square, numbered from the bottom.



■ **Figure 5** Partition of  $U^{p,B}$ , with  $p = 32$ , and the sets  $Level(\alpha)$  and  $Row(\alpha)$  for  $\alpha = 14$ .

► **Definition 11.** For all  $t = 0, \dots, \log_2 p - 1$ ,  $\ell = 0, \dots, t - 1$  and  $j = 1, \dots, 2^{t-\ell} - 1$  denote by  $R(t, \ell, j)$  the submatrix of matrix  $U^{p,B}$  of size  $2^\ell \times 2^\ell$  with left bottom corner in  $(p - 2^t - 2^\ell j - 1, p - 2^{\ell+1} + 1)$ .

$$R(t, \ell, j) = U[p - 2^t - 2^\ell(j + 1), \dots, p - 2^t - 2^\ell j - 1][p - 2^{\ell+1} + 1, \dots, p - 2^\ell]$$

Here, if  $j = 1$ , then all the entries below the main diagonal in matrix  $R(t, \ell, j)$  are set to zero.

In addition to that, for all  $t = 0, \dots, \log_2 p - 1$  denote by  $R(t, t, 0)$  the submatrix of matrix  $U^{p,B}$  of size  $2^t \times 2^t$  with left top corner at  $(p - 2^{t+1}, p - 2^t)$ , where all the entries above the main diagonal are set to zero. If  $0 \leq a, b < 2^t$  are the indices of this submatrix, then

$$R(t, t, 0)[a][b] = \begin{cases} U[p - 2^{t+1} + a][p - 2^t + b], & a \geq b \\ 0, & a < b \end{cases}$$

The lower triangular part of the  $t$ -th R-square is  $R(t, t, 0)$ . An R-square is split into blocks of columns, with each  $\ell$ -th block from the right of width  $2^\ell$ . Then, a submatrix  $R(t, \ell, j)$  is the  $j$ -th submatrix down in the  $\ell$ -th block of columns. The bottom submatrix in each block of columns (one with  $j = 1$ ) is upper triangular. All these details are illustrated in the figure. Submatrices  $L(\cdot, \cdot)$  and  $R(\cdot, \cdot, \cdot)$  form a partition of the upper triangle of matrix  $U$ , which is the non-zero part of  $U$ .

For each  $\alpha = 0, \dots, p-1$  denote by  $Row(\alpha)$  the set of all submatrices in the partition, which do intersect with the  $(p-1-\alpha)$ -th row of  $U^{p,B}$  (that is, the  $(\alpha+1)$ -th row from the bottom). The product of each of these matrices by the corresponding stripe of  $W^{p,q,C,D}$  is required for the computation of  $(p-1-\alpha)$ -th row of  $U \cdot W$ .

► **Definition 12.** Let  $\alpha = 2^t + r$  with  $0 \leq r < 2^t$ . Let  $r$  be represented as a sequence of binary digits as  $r = \sum_{i=1}^k 2^{r_i}$ , where  $r_1 < \dots < r_k$  are the numbers of all positive bits. Then  $Row(\alpha)$  consists of  $k$  matrices  $L(t, \cdot)$  and  $\lfloor \log_2 r \rfloor + 2$  matrices  $R(t, \cdot, \cdot)$ . To be more precise,

$$Row(\alpha) = \left\{ L \left( t, \sum_{i=s}^k 2^{r_i} \right) \mid s = 1, \dots, k \right\} \cup \left\{ R \left( t, \ell, \left\lfloor \frac{r}{2^\ell} \right\rfloor \right) \mid \ell = 0, \dots, \lfloor \log_2 r \rfloor \right\} \cup \{R(t, t, 0)\}$$

Let us now denote by  $Level(\alpha)$  the set of all matrices from  $Row(\alpha)$  whose bottom row intersects with the  $(p-1-\alpha)$ -th row of  $U^{p,B}$ . All matrices from  $Level(\alpha)$  are to be multiplied by the corresponding stripes of  $W$  simultaneously at a certain level of the circuit.

► **Definition 13.** For each  $\alpha = 1, \dots, p-1$ , let  $\alpha = 2^t + r$  with  $0 \leq r < 2^t$ . If  $r > 0$ , then the set of matrices  $Level(\alpha)$  consists of the following matrices.

$$Level(\alpha) = \{L(t, r)\} \cup \left\{ R \left( t, \ell, \frac{r}{2^\ell} \right) \mid \ell = 0, \dots, \lfloor \log_2 r \rfloor, r : 2^\ell \right\}$$

And if  $r = 0$ , then  $Level(\alpha)$  contains only the big lower-triangular matrix  $R$ .

$$Level(\alpha) = \{R(t, t, 0)\}$$

In Figure 5, the sets  $Row(\alpha)$  and  $Level(\alpha)$  are illustrated for  $\alpha = 14$ .

- **Lemma 14.** Every matrix of the partition lies in exactly one set  $Level(\alpha)$ .
- **Lemma 15.** Every element of every matrix from  $Level(\alpha)$  has at most  $\alpha$  leaves.
- **Lemma 16.** Every matrix from  $Level(\alpha)$  is multiplied by some elements of  $W$ , each having at most  $\min(\alpha, q)$  leaves.
- **Lemma 17.** For every  $\alpha$ , every matrix from  $Row(\alpha)$  lies in set  $Level(\beta)$ , for some  $\beta \leq \alpha$ .

In other words, in the circuit, every matrix from the partition shall be multiplied by the corresponding stripe of  $W$  before their product would be needed.

### 4.3 Circuit

The circuit is organized into  $6 \lceil \log_{\frac{3}{2}} n \rceil$  levels, each of depth at most  $\log n$ . At each group of six levels numbered  $6t, 6t+1, \dots, 6t+5$ , all elements with  $\alpha$  in the range  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$  are computed. This is done in six stages, according to the following plan.

0. At the level  $\tau = 6t$ , elements  $A(\cdot, \cdot)$  with  $\alpha$  leaves, where  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$  are computed directly via summation. These elements depend on  $B, C$  and  $\frac{A}{E}$  with  $\beta < \left(\frac{3}{2}\right)^t$  leaves – each of which was computed at a level  $6t'$  or  $6t'+3$ , where  $t' < t$ .
1. At the level  $\tau = 6t+1$  for all such  $\alpha$  that  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$  matrices from  $Level^{p,B}(\alpha)$  are multiplied by the corresponding strips of the matrix  $W^{p,q,C,D}$ ; this is done separately for each  $p$ . As it was mentioned in Lemma 15, all elements of the matrix  $U^{p,B}$  involved in this computation have at most  $\alpha < \left(\frac{3}{2}\right)^{t+1}$  leaves, so these elements were computed at levels  $6t'$  with  $t' \leq t$ . At the same time, as it follows from Lemma 16, the elements

of  $W^{p,q,C,D}$  involved in the computation have at most  $\min(q, \alpha) < \left(\frac{3}{2}\right)^t$  leaves, so these elements were computed at levels  $6t' + 3$  with  $t' < t$ . All in all, the elements computed at level  $\tau = 6t + 1$  depend only on elements from earlier levels.

2. At the level  $\tau = 6t + 2$ , elements  $\frac{E}{D}|_q(\cdot, \cdot, \cdot, \cdot)$  with  $q < \left(\frac{3}{2}\right)^t$  and with  $\alpha$  leaves, where  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$ , are computed. For each  $\alpha$ , the earlier computed products of matrices from the set  $Row^{p,B}(\alpha)$  by the corresponding stripes of  $W^{p,q,C,D}$  are taken, all their  $\alpha$ -th rows are summed up. Next, a sum of the resulting rows is taken over all rules  $E \rightarrow BC$ . Since submatrices from  $Row^{p,B}(\alpha)$  form a partition of the non-zero part of  $\alpha$ -th row of matrix  $U^{p,B}$ , the sum is exactly the value of  $\frac{E}{D}|_q$ . As it was proven in Lemma 17, each matrix from  $Row^{p,B}(\alpha)$  lies in one of the sets  $Level^{p,B}(\beta)$  with  $\beta \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$ . Hence, the elements from level  $\tau = 6t + 2$  depend on elements from levels  $6t' + 1$ , with  $t' \leq t$ .
3. At the level  $\tau = 6t + 3$ , according to the formulas from Section 4.1, elements  $\frac{A}{D}(\cdot, \cdot, \cdot, \cdot)$  with  $\alpha$  leaves are computed, where  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$ . To be more precise, for all  $\alpha$  in this range and for all  $\zeta = 1 \dots n - \alpha$ , the following matrix products are computed simultaneously:

$$\left\{ \frac{A}{D}[I][K] \right\}_{\substack{|I|=\zeta+\alpha \\ |K|=\zeta}} = \sum_{\substack{E \in N \\ \gamma: \frac{2}{3}\alpha + \zeta < \gamma < \zeta + \alpha}} \left\{ \frac{A}{E}[I][S] \right\}_{\substack{|I|=\zeta+\alpha \\ |S|=\gamma}} \cdot \left\{ \frac{E}{D} \Big|_{\frac{2}{3}\alpha} [S][K] \right\}_{\substack{|S|=\gamma \\ |K|=\zeta}}$$

In the left part of the equation there are exactly the elements  $\frac{A}{D}$  with  $\alpha$  leaves. They depend on elements  $\frac{A}{E}$  with at most  $\frac{1}{3}\alpha < \left(\frac{3}{2}\right)^t$  leaves, which were computed at some levels  $6t' + 3$  with  $t' < t$ . Also there is a dependency on elements  $\frac{E}{D}|_q$  with fewer than  $\alpha < \left(\frac{3}{2}\right)^{t+1}$  leaves, where  $q = \frac{2}{3}\alpha < \left(\frac{3}{2}\right)^t$ . These elements were computed at levels  $\{6t' + 2 | t' \leq t\} \cup \{6t' + 5 | t' < t\}$ . Overall, the elements from level  $\tau = 6t + 3$  depend on elements from levels  $6t' + 3$  and  $6t' + 5$  with  $t' < t$ , as well as on elements from levels  $6t' + 2$  with  $t' \leq t$ .

4. At the level  $\tau = 6t + 4$ , matrices from the sets  $Level^{p,B}(\alpha)$ , with  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$ , are multiplied by the corresponding strips of matrices  $W^{p,q,C,D}$ , and for all  $q \geq \left(\frac{3}{2}\right)^t$ . The computations are the same as at level  $6t + 1$ , with the only difference that  $\min(q, \alpha) < \left(\frac{3}{2}\right)^{t+1}$ . So, the elements from level  $\tau = 6t + 4$  depend on elements from levels  $6t'$  and  $6t' + 3$  with  $t' \leq t$ .
5. At the level  $\tau = 6t + 5$ , the elements  $\frac{E}{D}|_q$  with  $q \geq \left(\frac{3}{2}\right)^t$  are computed for all  $\alpha$  with  $\left(\frac{3}{2}\right)^t \leq \alpha < \left(\frac{3}{2}\right)^{t+1}$ . This is done similarly to the level  $6t + 2$ , with the computations depending on the elements from levels  $6t' + 4$  with  $t' \leq t$ .

► **Theorem 18.** *For every grammar  $G = (\Sigma, N, R, S)$  in the Chomsky normal form and for every string length  $n$ , there exists an arithmetic circuit of depth  $O((\log n + \log |R|) \log n)$  and with  $O(|N|^2 \cdot |R| \cdot n^{\omega+3})$  elements, which computes the number of parse trees of an input string of length  $n$ . (where  $\omega \geq 2$  is a number, such that for every  $n$  there is an arithmetic circuit of depth  $O(\log n)$  and with  $O(n^\omega)$  elements that computes the product of two  $n \times n$  integer matrices)*

**Proof.**

**Depth.** Since the elements at every level  $\tau$  depend only on elements from earlier levels, there are no cycles in the circuit, and the depth of the logical dependencies between elements does not exceed the number of levels, which is  $6\lceil \log_{\frac{3}{2}} n \rceil$ . At each level, each element is computed either as a sum of at most  $n^3|R|$  values, or as a part of a matrix product involving matrices of

size at most  $n \times n$ . Sums are computed in depth  $O(\log n + \log |R|)$  by the standard method. Matrix products are computed in depth  $O(\log n)$  using fast matrix multiplication. Hence, the entire circuit has depth  $O((\log n + \log |R|) \log n)$ .

**Number of elements.** Elements  $B(\cdot, \cdot)$  are computed as sums of  $O(|N| \cdot |R| \cdot n^5)$  elements, which is less than  $O(|N|^2 \cdot |R| \cdot n^{3+\omega})$ .

The computations of all elements  $\frac{A}{D}(\cdot, \cdot, \cdot, \cdot)$  are performed at levels  $6t + 3$ , see stage 3 above. For fixed  $\alpha$  and  $\zeta$ , there are  $\alpha + \zeta - (\frac{2}{3}\alpha + \zeta) = \frac{1}{3}\alpha$  different of  $\gamma$ , and for each  $\gamma$  there are  $|N|^3$  matrix multiplications corresponding to different nonterminals  $A$ ,  $D$  and  $E$ , where each matrix is not larger than  $n \times n$ . Summing up over all  $\alpha$ ,  $\zeta$ ,  $A$ ,  $D$  and  $E$  gives the following upper bound on the number of operations.

$$\text{const} \cdot \sum_{A, D, E \in N} \sum_{\zeta=1}^{n-1} \sum_{\alpha=1}^{n-\zeta} \frac{1}{3} \alpha n^\omega = \text{const} \cdot |N|^3 \cdot n(n^2 - 1)n^\omega = O(|N|^3 \cdot n^{3+\omega})$$

It remains to prove that all matrix multiplications required to compute  $\frac{E}{D}|_q(\cdot, \cdot, \cdot, \cdot)$  can be done using  $O(|N|^2 \cdot |R| \cdot n^{3+\omega})$  elements.

Let firstly fix the parameters  $p, q$ , the rule  $E \rightarrow BC$  and the nonterminal  $D$ , and estimate the number of elements used to compute the product of matrices  $U^{p,B} \cdot W^{p,q,C,D}$  for these  $p, q$ ,

Denote the width of the matrix  $W^{p,q,C,D}$  by  $M$ , it is  $O(n^2)$ . Fix  $t \in \{0, \dots, \log_2 p\}$  and consider the partition of matrices  $U^{p,B}[p - 2^{t+1}, \dots, p - 2^t - 1][p - 2^{t+1}, \dots, p - 2^t - 1]$  (the large square of size  $2^t \times 2^t$  on the main diagonal in Figure 5) and  $U^{p,B}[p - 2^{t+1}, \dots, p - 2^t - 1][p - 2^t, \dots, p - 1]$  (the large square of size  $2^t \times 2^t$  to the right of the main diagonal in Figure 5). The first matrix has  $2^{k-j-1}$  submatrices  $L(t, \cdot)$  of size  $2^j \times 2^j$ , and the second one has  $2^{k-j} - 1$  submatrices  $R(t, j, \cdot)$  of size  $2^j \times 2^j$ . So, among the matrices  $\{L(t, \cdot)\} \cup \{R(t, \cdot, k) \mid k > 0\}$  there are  $2^{t-j-1} + 2^{t-j} - 1$  matrices of size  $2^j \times 2^j$ , and each of them is multiplied by a strip of size  $2^t \times M$  of the matrix  $W^{p,q,C,D}$ ; to be more precise, this product is computed by splitting the stripe into  $\frac{M}{2^j}$  square matrices of size  $2^t \times 2^t$  each. Each product of square matrices of size  $2^j \times 2^j$  is computed using  $O(2^{j\omega})$  arithmetic operations, and therefore the total number of operations is bounded by constant times the following expression.

$$\begin{aligned} \sum_{j=0}^{t-1} (2^{t-j} + 2^{t-j-1} - 1) \frac{M}{2^j} 2^{j\omega} &\leq \sum_{j=0}^{t-1} \frac{3}{2} 2^{t-j} \frac{M}{2^j} 2^{j\omega} = \frac{3}{2} M \sum_{j=0}^{t-1} 2^{t-j} 2^{j\omega-j} = \\ &= \frac{3}{2} 2^t M \sum_{j=0}^{t-1} 2^{j(\omega-2)} \approx \text{const} \cdot 2^t M \cdot 2^{t(\omega-2)} = \text{const} \cdot 2^{t(\omega-1)} M \end{aligned}$$

In addition to that, there is one more matrix  $R(t, t, 0)$  of size  $2^t \times 2^t$ , which is multiplied by the strip of the matrix  $W^{p,q,C,D}$  of size  $2^t \times M$ , implemented as  $\frac{M}{2^t}$  multiplications by a square matrix, which gives  $O(2^{t\omega}) \cdot \frac{M}{2^t} = O(M2^{t(\omega-1)})$  additional operations.

All in all,  $O(M2^{t(\omega-1)})$  elements are required for a given  $t$ . The sum of the number of elements for  $t = 0, \dots, \log_2 p$  is then bounded by

$$\text{const} \cdot \sum_{t=0}^{\log p} M \cdot 2^{t(\omega-1)} \approx \text{const} \cdot M \cdot 2^{(\omega-1) \log p} = \text{const} \cdot p^{\omega-1} M$$

Hence, for a given  $p$  and  $q$  all matrices from the partition of  $U^{p,B}$  are multiplied using  $O(Mp^{\omega-1})$  elements, where  $M = O(n^2)$ . Since  $p \leq n$ , this value is not more than  $O(n^{\omega+1})$ .

The last step is to sum the number of elements for all  $p, q = 1, \dots, n$  and for all rules  $E \rightarrow BC$  and nonterminals  $D$ , and obtain the desired upper bound  $O(|N| \cdot |R| \cdot n^{\omega+3})$  on the number of elements in the circuit. ◀

Like the simple circuit with  $O(n^6)$  elements, the new circuit can also be adapted to compute the value of a string in a weighted grammar. Since fast matrix multiplication requires matrices over a ring, this restriction makes its way into the theorem.

► **Theorem 19.** *For every weighted grammar  $G = (\Sigma, N, R, S)$  in the Chomsky normal form, with weights in a ring  $\mathcal{R}$ , there is a uniform family of arithmetic circuits computing elements over  $\mathcal{R}$ , for each length of input strings  $n \geq 1$ , which computes the value of an input string of length  $n$  in the ring, has  $O(|N|^2 \cdot |R| \cdot n^{\omega+3})$  elements and is of depth  $O((\log n + \log |R|) \log n)$ . (where  $\omega \geq 2$  is a number, such that for every  $n$  there is a an arithmetic circuit of depth  $O(\log n)$  and with  $O(n^\omega)$  elements that computes the product of two  $n \times n$  matrices over  $\mathcal{R}$ )*

## 5 Future work

The smaller of the two proposed circuits for enumerating parse trees uses  $O(n^{3+\omega})$  elements. On the other hand, in the Boolean case, where only the existence of a parse tree is determined, it should be sufficient to use only  $O(n^{2\omega})$  elements, see Brent and Goldschlager [3, Sect. 6]. Perhaps the enumeration of parse trees might also be done using asymptotically fewer than  $n^{\omega+3}$  elements, and this looks like an interesting subject for future research.

---

## References

- 1 Ekaterina Bakinova, Artem Basharin, Igor Batmanov, Konstantin Lyubort, Alexander Okhotin, and Elizaveta Sazhneva. Formal languages over GF(2). *Inf. Comput.*, 283:104672, 2022. doi:10.1016/j.ic.2020.104672.
- 2 José-Miguel Benedí and Joan-Andreu Sánchez. Fast stochastic context-free parsing: A stochastic version of the Valiant algorithm. In Joan Martí, José-Miguel Benedí, Ana Maria Mendonça, and Joan Serrat, editors, *Pattern Recognition and Image Analysis, Third Iberian Conference, IbPRIA 2007, Girona, Spain, June 6-8, 2007, Proceedings, Part I*, volume 4477 of *Lecture Notes in Computer Science*, pages 80–88. Springer, 2007. doi:10.1007/978-3-540-72847-4\_12.
- 3 Richard P. Brent and Leslie M. Goldschlager. A parallel algorithm for context-free parsing. *Australian Computer Science Communications*, 6(7):7.1–7.10, 1984.
- 4 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.
- 5 Hillel Gazit and Gary L. Miller. An improved parallel algorithm that computes the BFS numbering of a directed graph. *Inf. Process. Lett.*, 28(2):61–65, 1988. doi:10.1016/0020-0190(88)90164-0.
- 6 Alexander Okhotin. Parsing by matrix multiplication generalized to Boolean grammars. *Theor. Comput. Sci.*, 516:101–120, 2014. doi:10.1016/j.tcs.2013.09.011.
- 7 Wojciech Rytter. Parallel time  $o(\log n)$  recognition of unambiguous cfs. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 380–389. Springer, 1985. doi:10.1007/BFb0028822.
- 8 Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
- 9 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

# Spartan Bipartite Graphs Are Essentially Elementary

Neeldhara Misra   

Department of Computer Science and Engineering, Indian Institute of Technology, Gandhinagar, India

Saraswati Girish Nanoti 

Department of Mathematics, Indian Institute of Technology, Gandhinagar, India

---

## Abstract

---

We study a two-player game on a graph between an attacker and a defender. To begin with, the defender places guards on a subset of vertices. In each move, the attacker attacks an edge. The defender must move at least one guard across the attacked edge to defend the attack. The defender wins if and only if the defender can defend an infinite sequence of attacks. The smallest number of guards with which the defender has a winning strategy is called the eternal vertex cover number of a graph  $G$  and is denoted by  $\text{evc}(G)$ . It is clear that  $\text{evc}(G)$  is *at least*  $\text{mvc}(G)$ , the size of a minimum vertex cover of  $G$ . We say that  $G$  is *Spartan* if  $\text{evc}(G) = \text{mvc}(G)$ . The characterization of Spartan graphs has been largely open. In the setting of bipartite graphs on  $2n$  vertices where every edge belongs to a perfect matching, an easy strategy is to have  $n$  guards that always move along perfect matchings in response to attacks. We show that these are essentially the only Spartan bipartite graphs.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Graph theory; Theory of computation  $\rightarrow$  Algorithm design techniques

**Keywords and phrases** Bipartite Graphs, Eternal Vertex Cover, Perfect Matchings, Elementary, Spartan

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.68

**Related Version** *Full Version:* <https://arxiv.org/abs/2308.04548>

**Funding** Both authors acknowledge support from IIT Gandhinagar. Further, the first author acknowledges support from the SERB grant ECR/2018/002967 and the second author acknowledges support from CSIR.

**Acknowledgements** The authors are grateful to several anonymous reviewers for helpful comments on earlier versions of this manuscript.

## 1 Introduction

A *vertex cover* of a graph  $G$  is a subset  $S$  of vertices such that every edge in  $G$  has at least one of its endpoints in  $S$ . An *optimal* vertex cover of a graph  $G$  is a vertex cover of the smallest possible size and the size of this optimal vertex cover is denoted by  $\text{mvc}(G)$ . A *bipartite graph*  $G = (V, E)$  is a graph whose vertex set can be partitioned into two independent sets, say  $V = (A \cup B)$ , that is 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* optimal 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*.

Klostermeyer and Mynhardt [8] introduced the notion of the *eternal vertex cover* number of a graph  $G$ . There are two players – an *attacker* and a *defender*, who are playing on a simple, undirected graph  $G$ . In the beginning, the defender can choose to place guards on some of the vertices of  $G$ . The attacker’s move involves choosing an edge to “attack”. The defender is able to “defend” this attack if she can move the guards along the edges of the graph in such a way that at least one guard moves along the attacked edge. If both the



© Neeldhara Misra and Saraswati Girish Nanoti;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 68; pp. 68:1–68:15

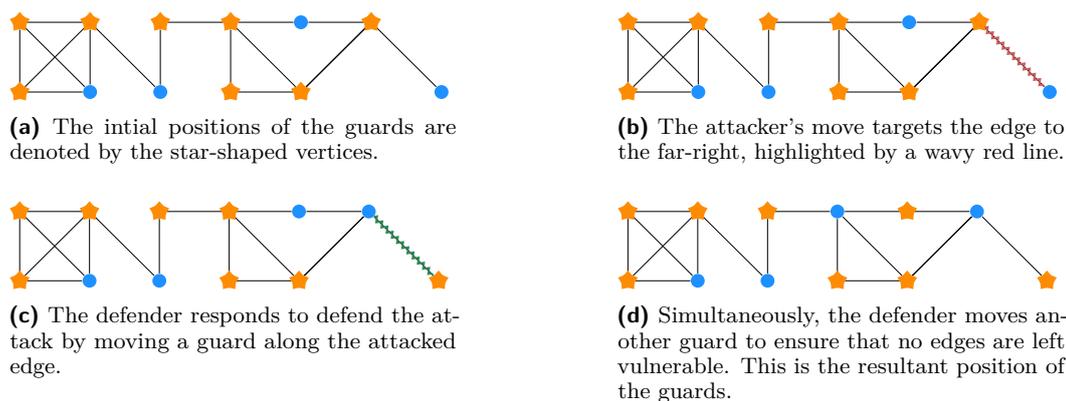
Leibniz International Proceedings in Informatics



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

endpoints of the attacked edge have guards on them, the guards can simply exchange their places and the attack is defended as well as the original configuration of guards is restored. Note that any number of guards can move after an attack, but each guard can move only one “step”, i.e., just along a single edge. If such a movement is not possible, then the attacker wins. If the defender can defend the graph against an infinite sequence of attacks, then the defender wins (see Figure 1). The minimum number of guards with which the defender has a winning strategy is called the *eternal vertex cover number* of the graph  $G$  and is denoted by  $\text{evc}(G)$ .

There are two variations of this game, one where only one guard can be present per vertex at a given time and a variant where more than one guard can be present on a vertex at a given time. Our results apply to both the variants of the game.



■ **Figure 1** An attack that is defended by moving two guards.

If  $S_\ell$  is the subset of vertices that have guards on them after the defender has played her  $\ell$ -th move, and  $S_\ell$  is not a vertex cover of  $G$ , then the attacker can target any of the uncovered edges to win the game. Therefore, when the defender has a winning strategy, it implies that she can always “reconfigure” one vertex cover into another in response to any attack, where the reconfiguration is constrained by the rules of how the guards can move and the requirement that at least one of these guards needs to move along the attacked edge. It follows that  $\text{evc}(G) \geq \text{mvc}(G)$ , where  $\text{mvc}(G)$  is the size of the smallest vertex cover for any graph  $G$ .

We call graphs  $G$  which enjoy  $\text{evc}(G) = \text{mvc}(G)$  *Spartan*, to indicate that these graphs can manage attacks without using “additional” guards. Understanding Spartan graphs for several special cases have been addressed in the literature (see, for instance, [3]). However, the characterization of bipartite graphs that are Spartan has been left open. We show that bipartite graphs are Spartan if and only if they are essentially elementary.

We also consider a natural extension of the eternal vertex cover problem: where the guards are allowed to move two or more steps on each turn instead of one, without retracing. An attack is defended only if the guard moves along the attacked edge on their first step. We show that with just one additional step, we need either just as many guards as the size of the smallest vertex cover or one more: so in this variant all graphs are “almost” Spartan. We also show that the extra guard is needed only if the graph has a degree one vertex not contained in any minimum vertex cover. Finally, we show that allowing for more than one extra step is the same as allowing exactly one extra step.

**Related Work.** Among characterizations related to Spartan graphs, we have the following. 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 Babu et al. [2] showed that  $G$  is Spartan if and only if for every vertex  $v \in V \setminus X$ , there exists a minimum vertex cover  $S_v$  of  $G$  such that  $X \cup \{v\} \subset S_v$ . Klostermeyer and Mynhardt [7] also study graphs for which the eternal vertex cover number coincides with the eternal domination number, a closely related notion. This is a similar game, except that the attacks happen on vertices and if the attacked vertex does not have a guard already, then a guard from a neighbouring vertex must come to the attacked vertex. All other guards can stay on their initial position or move to a neighbouring vertex. The minimum number of guards required to protect the graph from an infinite sequence of attacks is called the *eternal domination number*.

Note that twice as many vertices as the  $\text{mvc}(G)$  always suffice to defend against any sequence of attacks – by placing guards on both endpoints of any maximum matching to begin with and after any attack, reconfiguring the guards to obtain another maximum matching. Using this strategy, a 2-approximation algorithm for ETERNAL VERTEX COVER was obtained by Fomin et al. [5]. This also implies  $\text{mvc}(G) \leq \text{evc}(G) \leq 2\text{mvc}(G)$ . [8] gave a characterization of the graphs for which the upper bound is achieved. The notion of elementary graphs was considered by Lovász and Plummer [9] and several useful characterizations were given by Heteyi [6]. We adapt these definitions to suit the context of bipartite graphs.

## Methodology

Recall that a *matching* is a collection of vertex disjoint edges, and a matching which contains one edge incident to each vertex of a graph is called a *perfect matching*. We say that an edge is *allowed* if it is contained in some perfect matching. A graph  $G$  is said to be *elementary* if and only if it is connected and every edge is allowed.

It turns out that a connected bipartite graph  $G = (A \cup B, E)$  is elementary if and only if its *only* optimal vertex covers are  $A$  and  $B$ , as we make explicit in the proposition below.

► **Proposition 1** ([6]). *The following are equivalent for any connected bipartite graph  $G = (A \cup B, E)$ .*

1.  $A$  and  $B$  are the only minimum vertex covers of  $G$ , and in particular  $|A| = |B|$ .
2. Every edge in  $G$  is allowed.

Notice that it is easy to see that for bipartite graphs, if  $G$  is elementary then it is Spartan (see also Lemma 2 and Araki, Fujito, and Inoue [1]). Indeed, we start by placing guards on all vertices of  $A$ . If the edge  $e$  is attacked, then we move guards to  $B$  along the edges of the perfect matching  $M$  that contains the edge  $e$ . Future attacks can be similarly defended, so the guards alternate between occupying  $A$  and  $B$  in response to edge attacks.

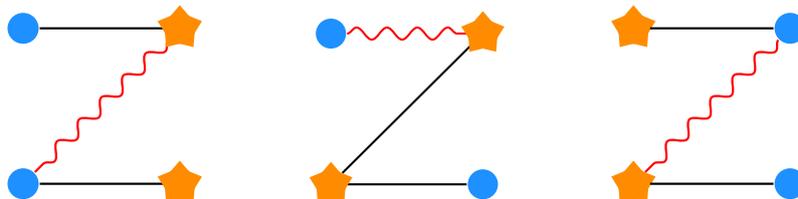
What we focus on demonstrating in this contribution is the converse, namely that if a connected bipartite graph  $G$  does not have this structure, then it is also not Spartan: in some sense, the trivial scenario is the only one in which we can hope to manage without employing any “extra” guards.

**Spartan  $\implies$  Perfect Matching.** To begin with, notice that if  $G = (A \cup B, E)$  is connected and does not have a perfect matching, we already need more than  $\text{mvc}(G)$  guards (see also Proposition 4). In particular, let  $M$  be a maximum matching in  $G$ . Recall that  $|M| = \text{mvc}(G)$  by König’s theorem. Without loss of generality, let  $b \in B$  be a vertex not incident to any edge of  $M$ . Since  $G$  is connected,  $b$  has some neighbor  $a \in A$ , which must belong to any vertex cover of  $G$ . In particular, the defender is forced to position a guard on

## 68:4 Spartan Bipartite Graphs Are Essentially Elementary

$a$  in the initial configuration. If the edge  $ab$  is attacked, the guard on  $a$  is forced to move to  $b$ . This creates a situation where  $|M| - 1$  guards have to reposition themselves to protect all the edges of  $M$  – which is impossible since the edges of  $M$  are disjoint.

Therefore, a necessary condition for a connected bipartite graph to be Spartan is that it must admit a perfect matching. This is, however, evidently not sufficient: indeed, there are connected bipartite graphs with perfect matchings that are not Spartan (see also Figure 2).

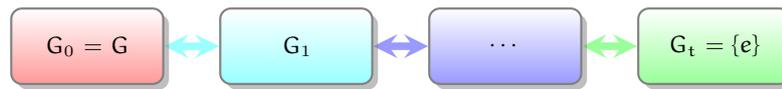


■ **Figure 2** A graph with a perfect matching where  $\text{mvc}(G)$  many guards do not suffice. For each choice of  $\text{mvc}(G)$  many guards at the initial configuration, attacking the dashed edge shown in the figure leads to an indefensible position in one or two steps.

**Spartan  $\implies$  No Degree One Vertices.** We observe that a connected bipartite Spartan graph  $G = (A \cup B, E)$  on at least three vertices cannot have a degree one vertex (see also Proposition 5 and Babu et al. [2]). Assume to the contrary that  $G$  has a degree one vertex, say  $a \in A$ . Let  $b$  denote  $a$ 's unique neighbor in  $G$ . Since  $G$  has at least one vertex other than  $a$  and  $b$ , and  $G$  is connected,  $b$  must have another neighbor in  $G$ : say  $a'$ . Now, if the initial configuration has a guard on  $a$ , then attack the edge  $ba'$ ; otherwise attack the edge  $ab$  to force a guard on  $a$  and then attack  $ba'$ . Both cases lead to a scenario where both endpoints of the edge  $ab$  have guards on them. However, since  $G$  is Spartan, it has a perfect matching  $M$ . Further, since  $a$  is a degree one vertex,  $M$  is forced to contain the edge  $ab$ . This in turn implies that at least  $|A| - 1$  guards are required to defend the rest of the graph, contradicting the assumption that  $G$  is Spartan.

**Proof by Structure-Preserving Contractions.** So far, we have seen that a connected bipartite graph  $G = (A \cup B, E)$  which is Spartan must have a perfect matching and no degree one vertices. Fix an arbitrary perfect matching  $M$ , and let  $ab \in M$ . Let  $P$  be a maximal walk (without repeating edges) starting at  $a$ , alternating between edges of  $M$  and  $E \setminus M$ . The terminal vertex of  $P$  is either  $a$ , or a vertex  $v$  which must have a neighbor on  $P$ , since  $G$  has no degree one vertices. So from  $P$ , we can derive a cycle whose edges alternate between  $M$  and  $E \setminus M$  (see also Proposition 17).

At a high level (see Figure 3), our proof relies on “contracting” each such cycle to a single matching edge and arguing that the resulting graph remains Spartan provided the original graph was also Spartan. In particular, we show the following. If  $G$  is Spartan to begin with, and  $M$  is a perfect matching with  $S$  is a set of endpoints of  $M' \subseteq M$  such that  $G[S]$  is elementary, then “replacing”  $G[S]$  with a single edge until this operation is no longer possible; keeps the graph Spartan (see also Lemma 13). Here it is useful that any bipartite graph obtained by adding edges to a cycle (while preserving the bipartite-ness) is elementary, a fact that we establish separately (see also Proposition 7) – thus we can simply keep contracting along cycles  $C$ . The idea of the proof is then the following. Consider what happens once we are stuck, i.e., we cannot contract any further: we either have an edge, or we have a graph that is not an edge but must have a degree one vertex (if not, note that we would not



■ **Figure 3** Proof idea: The sequence of forward contractions preserves “Spartan-ness” overall (i.e.,  $G_t$  is Spartan if  $G_0$  is Spartan) while the backward expansions preserve “elementary-ness”.

be stuck). But from our previous discussion, we know that a connected bipartite Spartan graph on at least three vertices cannot have a degree one vertex. So in this situation, we have transformed the Spartan graph we started with into one that is not Spartan, and in particular, there is an attack that  $\text{mvc}(\cdot)$  many guards will not be able to defend on the “contracted” graph. It turns out that this attack can be mimicked on the graph we started with, contradicting our assumption that the original graph is Spartan. Thus, our process must end at a single edge.

To conclude the proof, we show that if we take an elementary bipartite graph  $G$  and “inject” an elementary graph into it by substituting an edge with an elementary graph, then the resulting graph is also elementary. This lets us run the contraction operations on  $G$  in reverse, starting from the edge that we ended up with; and since an edge is elementary, we conclude that  $G$  must have also been elementary (see also Lemma 15).

When  $G$  is not connected, the arguments we made can be made independently of every connected component of  $G$ . This leads to our final characterization: that a bipartite graph  $G$  is Spartan if and only if each of its components are elementary. Since it is straightforward to check if a bipartite graph  $G = (A \cup B, E)$  is elementary (for instance, by checking if  $G \setminus \{u, v\}$  admits a perfect matching for every  $uv \in E$ ), our result also implies a polynomial-time recognition algorithm for bipartite Spartan graphs. We note that, in contrast to  $\text{mvc}(G)$ , determining  $\text{evc}(G)$  is known to be NP-hard even for bipartite graphs.

In Section 2, we describe the proofs of the claims mentioned in the outline here in greater detail. Our notation largely follows conventions from Diestel [4]. The proofs for statements marked with a  $(\star)$  can be found in the full version of this paper.

## 2 Spartan Bipartite Graphs are Essentially Elementary

We begin with the (easy) forward implication, which was also observed by [1] – we provide an explicit proof in the full version in the interest of completeness.

► **Lemma 2**  $(\star)$ . *If a bipartite graph  $G$  is essentially elementary,  $\text{evc}(G) = |V(G)|/2 = \text{mvc}(G)$ , i.e.,  $G$  is Spartan.*

It turns out that we may assume, without loss of generality, that  $G$  is connected (we refer the reader to the full version for a more explicit explanation). Therefore, our goal amounts to showing the following.

► **Lemma 3**  $(\star)$ . *If  $G$  is a connected bipartite graph that is Spartan, then it is elementary.*

We first argue that any connected Spartan bipartite graph must have a perfect matching.

► **Proposition 4**. *If  $G$  is a Spartan connected bipartite graph, it admits a perfect matching.*

**Proof.** Let  $G$  be a connected Spartan bipartite graph with bipartition  $(A, B)$  where  $A = \{a_1, \dots, a_p\}$  and  $B = \{b_1, \dots, b_q\}$  (see also Figure 4). For the sake of contradiction, assume that  $G$  does *not* admit a perfect matching, i.e., without loss of generality let  $q > k$ . Let  $S$  be

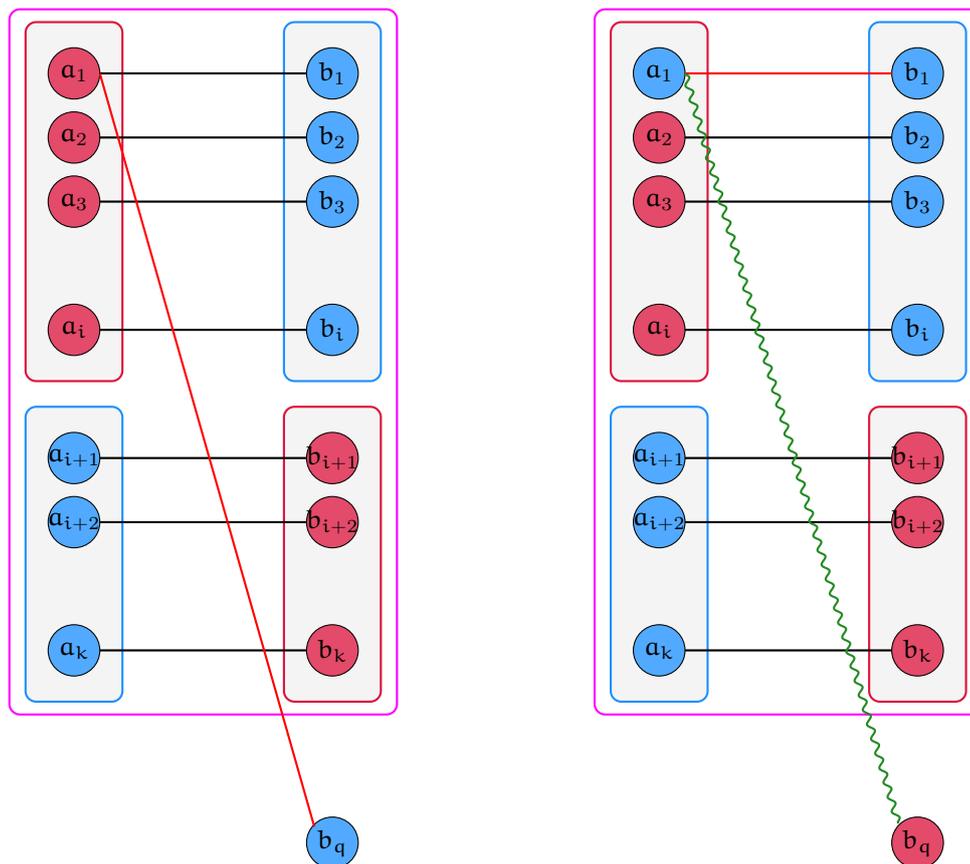
68:6 Spartan Bipartite Graphs Are Essentially Elementary

a minimum vertex cover in  $G$  of size  $k$  such that the guards are placed on  $S$  in the initial configuration. Let  $M$  be any maximum matching. Without loss of generality, let the edges of  $M$  be given by:

$$\{e_1 = (a_1, b_1), \dots, e_i = (a_i, b_i), e_{i+1} = (a_{i+1}, b_{i+1}), \dots, e_k = (a_k, b_k)\},$$

where  $S \cap A = \{a_1, \dots, a_i\}$  and  $S \cap B = \{b_{i+1}, \dots, b_k\}$ .

Since  $M$  is not a perfect matching,  $|V(G)| > 2k$  i.e. there exists a vertex in  $G$  which is not an endpoint of any edge in  $M$ . Without loss of generality, let this vertex be  $b_q$ . Since  $b_q$  is not an isolated vertex, it must have a neighbour on the  $A$  side. Since  $S$  is a vertex cover, this neighbour must lie in  $S \cap A$ . Without loss of generality, let this neighbour be  $a_1$ . If the attacker attacks  $a_1 b_q$ , the guard on  $a_1$  must come to  $b_q$  and cannot move further. Now there are  $k - 1$  guards on the vertices of  $M$  to protect  $k$  matching edges. So no matter how these guards arrange themselves, at least one edge of  $M$  must be vulnerable before the next attack. The attacker attacks this edge in the next move and wins thus contradicting the fact that  $G$  is Spartan. Therefore, the graph  $G$  must admit a perfect matching. ◀



(a) A graph which does not have a perfect matching and the red edge is attacked.

(b) After defending the attack, there are only  $k - 1$  guards to protect  $k$  matching edges. So an edge will remain vulnerable.

■ **Figure 4** Demonstrating that Spartan connected bipartite graphs must have perfect matching towards a proof of Proposition 4.

So we now have that connected Spartan bipartite graphs  $G$  have at least one perfect matching. From now on we denote the bipartitions of  $G$  by  $A = \{a_1, a_2, \dots, a_n\}$  and  $B = \{b_1, b_2, \dots, b_n\}$  where each  $a_i$  and  $b_i$  have an edge between them so that we have a perfect matching  $M = \bigcup_{i=1}^n \{a_i b_i\}$ . Since  $G$  is connected, these bipartitions are unique upto permutations. We now show that connected bipartite graphs that are Spartan do not have degree one vertices. This follows from known results (for example, as shown by [2]), however, our argument is more direct and we make it explicit to make our presentation self-contained.

► **Proposition 5** (No Degree One Vertex). *Let  $G$  be a connected bipartite graph with  $|V(G)| > 2$  and with a perfect matching. If  $\text{evc}(G) = \text{mvc}(G)$ , then  $G$  has no degree one vertex.*

From Lemma 2 and Proposition 5, we have the following.

► **Corollary 6.** *Let  $G$  be a connected bipartite graph with more than two vertices. If  $G$  is elementary, then  $G$  cannot have a degree one vertex.*

Next, we show that a bipartite graph with a cycle that visits every vertex is elementary.

► **Proposition 7** (Adding edges to cycles). *Consider a connected bipartite graph  $G = (A \cup B, E)$  which is a cycle  $a_1 b_1 a_2 b_2 \dots a_k b_k a_1$  (where  $k > 1$ ). The graph  $G'$  formed after adding any number of edges between  $A$  and  $B$  (preserving bipartiteness) will be elementary.*

**Proof.** Let  $G$  be a connected bipartite graph  $G$  which is a cycle  $a_1 b_1 a_2 b_2 \dots a_k b_k a_1$ , where  $k > 1$  (see also Figure 5). Here the two sides of the bipartition are  $A = \{a_1, a_2, \dots, a_k\}$  and  $B = \{b_1, b_2, \dots, b_k\}$ .

The graph  $G$  has two perfect matchings which are given by  $M = \{a_1 b_1, a_2 b_2, \dots, a_k b_k\}$  and  $M' = \{b_1 a_2, b_2 a_3, \dots, b_{k-1} a_k, b_k a_1\}$ . It is easily seen that any edge of  $G$  lies in one of these two perfect matchings.

Suppose we add some edges from  $A$  to  $B$ . Notice that the vertex set of the new graph  $G'$  is same as the vertex set of  $G$ . In order to prove that  $G'$  is elementary, it is sufficient to show that each of the newly added edges belongs to a perfect matching of  $G'$ .

Consider a newly added edge  $a_i b_j$ . This clearly preserves bipartiteness. Here,  $i \neq j, j + 1$  because the edges  $a_i b_i$  and  $a_i b_{i-1}$  were already in  $G$ .

First, let  $i > j + 1$ . Consider the matching:

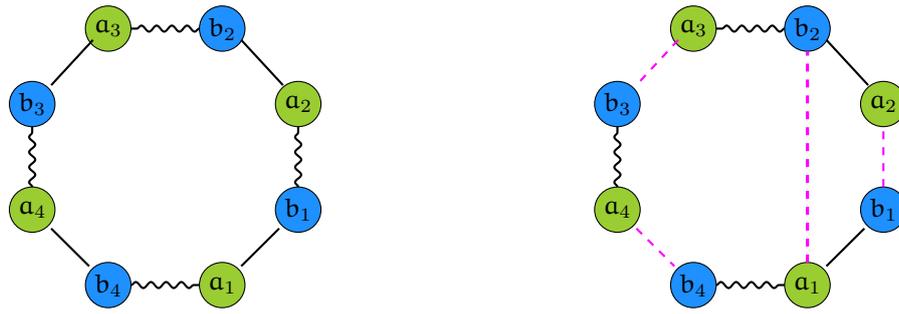
$$M_1 = \{a_i b_j, b_i a_{i+1}, b_{i+1} a_{i+2}, \dots, b_{k-1} a_k, b_k a_1, b_1 a_2, \dots, b_{j-1} a_j, a_{j+1} b_{j+1}, a_{j+2} b_{j+2}, \dots, a_{i-1} b_{i-1}\}.$$

Clearly  $M_1$  is a perfect matching containing  $a_i b_j$ .

Now let  $i < j$ . Let  $N_1 = \{a_i b_j, b_i a_{i+1}, b_{i+1} a_{i+2}, \dots, b_{j-1} a_j\}$ ;  $N_2 = \{a_1 b_1, \dots, a_{i-1} b_{i-1}\}$ ; and  $N_3 = \{a_{j+1} b_{j+1}, \dots, a_k b_k\}$ .

Consider the matching  $M_2$  given by  $N_1 \cup N_2 \cup N_3$ . Clearly  $M_2$  is a perfect matching containing  $a_i b_j$ .

This concludes our argument showing that  $G'$  is elementary. ◀



(a) This figure shows a cycle with  $k = 4$ .

(b) Each newly added edge is allowed: Suppose  $a_1 b_2$  is the new edge, it can be combined with the existing edges to form the perfect matching denoted by the dashed purple lines.

■ **Figure 5** An example of a bipartite graph with a spanning cycle.

We now introduce the terminology “special subset” to indicate that we are working with a subset of endpoints of some edges in a perfect matching.

► **Definition 8** (Special Subset). *Let  $G$  be a connected bipartite graph with a perfect matching  $M = \{a_1 b_1, a_2 b_2, \dots, a_n b_n\}$ . Consider:  $S = \{a_{i_1}, b_{i_1}, a_{i_2}, b_{i_2}, \dots, a_{i_k}, b_{i_k}\}$  for some distinct  $i_1, i_2, \dots, i_k \in [n]$  and  $2 \leq k \leq n$ . Then  $S$  is said to be a **special subset** of  $V(G)$ .*

A “special induced subgraph” is a subgraph induced by a special subset.

► **Definition 9** (Special Induced Subgraph). *Let  $G$  be a connected bipartite graph with a perfect matching and  $S$  be a special subset of  $V(G)$ . The subgraph  $G[S]$  induced by  $S$  is called a **special induced subgraph** of  $G$ .*

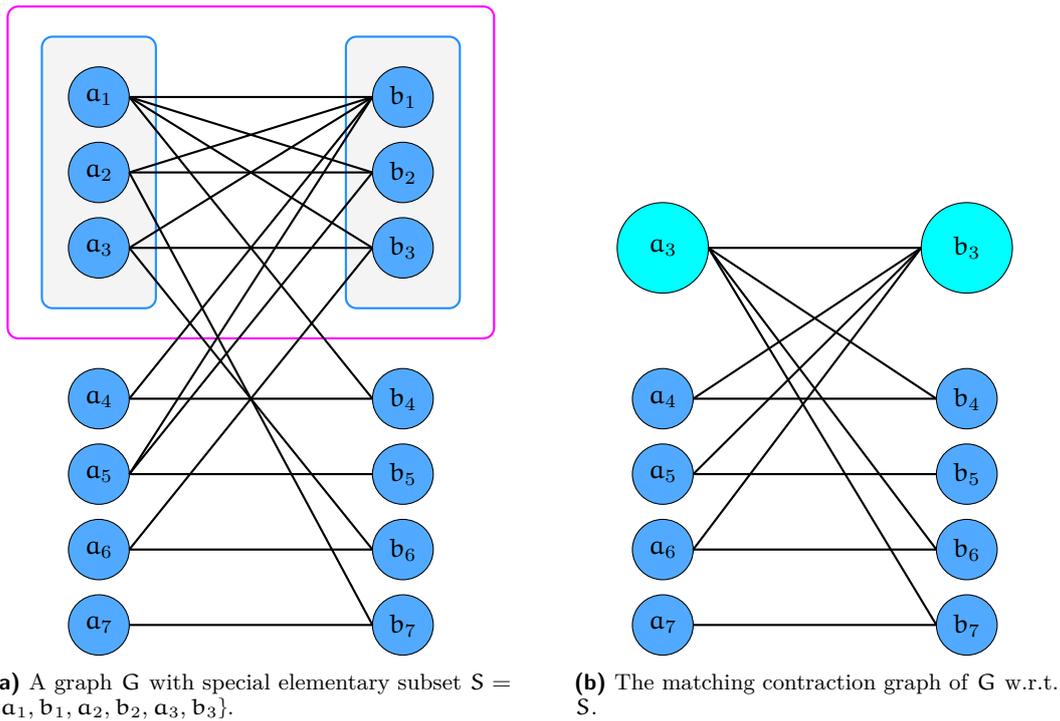
We now note that edges allowed in special induced subgraphs of a graph  $G$  are also allowed in  $G$ .

► **Proposition 10** (Allowed Edge). *Let  $G$  be a connected bipartite graph with a perfect matching  $M$ . Let  $S$  be a special subset of  $G$  and  $H = G[S]$  be the corresponding special induced subgraph. An edge which is allowed in  $H$  is also allowed in  $G$ .*

**Proof.** If  $S = V(G)$ , we are done. Otherwise without loss of generality, let  $S$  be the set  $\{a_1, b_1, a_2, b_2, \dots, a_k, b_k\}$ . Consider an edge  $a_i b_j$  which is allowed in  $H$  for some  $i, j \in [k]$ . Consider the matching  $M_H$  which is a perfect matching of  $H$  containing  $a_i b_j$ . Consider  $M' = M_H \cup \{a_{i+1} b_{i+1}, \dots, a_k b_k\}$ . Clearly  $M'$  is a perfect matching of  $G$  containing  $a_i b_j$ . Thus,  $a_i b_j$  is allowed in  $G$ . ◀

We now turn to a key definition: one of “contracting” a special elementary subset to a single edge. We refer the reader to Figure 6 for an example of an application of this operation.

► **Definition 11** (Matching Contraction Graph). *Consider a connected bipartite graph  $G$  with bipartition  $A$  and  $B$  and a perfect matching  $M$ . Let  $S$  and  $H = G[S]$  be a special subset of  $V(G)$  and the corresponding special induced subgraph of  $G$ . If  $H$  is elementary, replace  $A \cap S$  by a new vertex  $\alpha$  and  $B \cap S$  by a new vertex  $\beta$  in  $G$ . Add an edge  $\alpha\beta$  in the new graph. For an edge  $ab$  where  $a \in A \setminus S$  and  $b \in S$ , add an edge  $a\beta$  to the new graph. Similarly for an edge  $ab$  where  $a \in S$  and  $b \in B \setminus S$ , add an edge  $\alpha b$  to the new graph. Keep all the other edges and vertices of  $G$  unchanged. If there are any parallel edges, replace them by a single edge. The new graph  $G_S$  thus obtained is called the **matching contraction graph** of  $G$  w.r.t.  $S$  and the procedure of obtaining a matching contraction graph is called as **contraction**.*



■ **Figure 6** Demonstrating the operation of contracting a special elementary subset to a single edge. For a more elaborate example involving a sequence of contractions, see Figure 12.

Note that if  $G$  is a connected bipartite graph and  $S$  is a special subset, then the matching contraction graph  $G_S$  is also connected and bipartite.

It can be seen that if the special subset  $S$  has size  $2k$ , then the matching contraction graph has  $2(n - k + 1)$  vertices, a perfect matching of size  $n - k + 1$  and thus a minimum vertex cover of size  $n - k + 1$ .

► **Definition 12** (Special elementary subset). *Let  $G$  be a connected bipartite graph with a perfect matching. A special subset  $S$  such that  $G[S]$  is elementary is called a **special elementary subset**.*

We consider a series of contractions over a graph  $G$ . Each time we contract a special elementary subset  $S$  of  $G$ . We rename each vertex in the new graph as the largest indexed vertex in the original graph. We associate a label with each vertex, which carries the names of the vertices of the original graph that were contracted to this new vertex. We keep on repeating this procedure until we reach a graph which cannot be contracted further. We denote this graph by  $G_{\bar{S}}$ . We fix a perfect matching  $M$  in  $G$  and in each contracted graph, we get a matching given by  $M \setminus E(G[S]) \cup \{\alpha\beta\}$  where  $\alpha$  and  $\beta$  denote the newly obtained vertices on the  $A$  and  $B$  side respectively. Whenever we say “matched partner” or “matching edge” in the new graph, unless mentioned otherwise, we will be referring to this matching.

► **Lemma 13** (Matching Contraction Graph Property). *Suppose that  $G$  is a connected bipartite graph with a perfect matching  $M$ . Then, if  $G$  is Spartan, then  $G_{\bar{S}}$  is Spartan.*

**Proof.** If  $G_{\bar{S}}$  is a single edge, then  $G$  was elementary to begin with, as each expansion preserves the “elementary-ness” by Lemma 15. And since a single edge is Spartan,  $G_{\bar{S}}$  is Spartan in this case.

## 68:10 Spartan Bipartite Graphs Are Essentially Elementary

Consider the case where  $G_{\bar{s}}$  is not a single edge. Now by Proposition 18,  $G_{\bar{s}}$  must have a degree-1 vertex. Without loss of generality, we assume that this degree-1 vertex is on the B side and let us denote this vertex by  $\beta_1$  and its matched partner by  $\alpha_1$ . Also, note that  $G_{\bar{s}}$  is not Spartan by Proposition 5. We show that  $G$  was also not Spartan in this case.

Let  $B_1$  be the set of vertices in  $G$  in the label of  $\beta_1$ , i.e, which were contracted to get  $\beta_1$  and similarly  $A_1$  be the set of vertices in  $A$  which were contracted to get  $\alpha_1$  and let  $|A_1| = |B_1| = k$ . Note that this contraction may not have happened in one step, but over a series of steps. Notice that  $G \setminus \{A_1 \cup B_1\}$  is non-empty, as  $G_{\bar{s}}$  is not a single edge. Note that no vertex in  $B_1$  can be adjacent to a vertex in  $A \setminus A_1$ . Suppose some vertex  $b \in B_1$  is adjacent to a vertex  $a \in A \setminus A_1$ , this will result in an edge  $a\beta_1$  in  $G_{\bar{s}}$  which contradicts the fact that  $\beta_1$  is a degree-1 vertex. Also, we know that after a series of contractions,  $G[A_1 \cup B_1]$  got contracted to a single edge. By Lemma 15,  $G[A_1 \cup B_1]$  is elementary. Therefore, any minimum sized vertex cover of  $G$  will contain all the  $k$  vertices of  $A_1$  and no vertex from  $B_1$  or all the  $k$  vertices of  $B_1$  and no vertex from  $A_1$ . Now consider any initial configuration of guards on  $G$  with  $n$  guards. As seen above, either all the vertices of  $B_1$  have guards or all the vertices of  $A_1$  have guards and only one of these can happen. If all the vertices of  $A_1$  are occupied by guards, the attacker attacks a matched edge  $a_i b_i$  such that  $a_i \in A_1$ . The guard on  $a_i$  is forced to move to  $b_i$  and cannot move any further. If all the guards cannot reconfigure to form a vertex cover, then  $G$  is not Spartan and hence we are done. Otherwise, all the guards must reconfigure in such a way that all the vertices in  $B_1$  have guards and all the vertices in  $A_1$  do not have guards. Thus without loss of generality, we can assume that all the vertices in  $B_1$  have guards and all the vertices in  $A_1$  do not have guards.

Now recall that no vertex in  $B_1$  has a neighbour in  $A \setminus A_1$ . Since  $G$  is connected, there exists  $a_p \in A_1$  and  $b_q \in B \setminus B_1$  such that  $a_p b_q \in E(G)$ . Since  $a_p$  does not have a guard,  $b_q$  must have a guard as the guards are occupying a vertex cover. Now suppose the attacker attacks the edge  $a_p b_q$ , the guard on  $b_q$  must move to  $a_p$ . Since all the vertices in  $B_1$  have guards but none of them has a neighbour in  $A \setminus A_1$  and no vertex in  $A_1$  had a guard before this guard just moved to  $a_p$ , now no guard can move from  $\{A_1 \cup B_1\}$  to  $V(G) \setminus A_1$ . Thus there will be  $k + 1$  guards on  $k$  matching edges in  $G[A_1 \cup B_1]$ . Therefore, some edge in  $G[V(G) \setminus \{A_1 \cup B_1\}]$  will be vulnerable and can be attacked in the next move. Hence,  $G$  is not Spartan.  $\blacktriangleleft$

► **Remark.** We have actually shown that any bipartite graph  $G$  which is not “essentially elementary” can be destroyed by the attacker in at most three moves when the defender has  $\text{mvc}(G)$  many guards.

► **Definition 14 (Maximal Contraction Graph).** *Let  $G$  be a connected bipartite graph with a perfect matching and there does not exist any special elementary subset  $S$  of  $V(G)$ . Then  $G$  is said to be a **maximal contraction graph**.*

Note that by definition the size of a special subset is always more than 2 and hence if  $|V(G)| = 2$ , i.e.,  $G$  is a single edge, then  $G$  is a maximal contraction graph as  $G$  has no special subset and hence no special elementary subset. Note that the sequence of contractions above ends in a maximal contraction graph that is in fact an edge.

► **Lemma 15 (Maximal Contraction Graph).** *Let  $G$  be a connected bipartite graph with a perfect matching  $M$ . Let  $S$  be a special elementary subset of  $V(G)$  and  $G_S$  be the corresponding matching contraction graph. Then, if  $G_S$  is elementary, then  $G$  is also elementary.*

**Proof.** Let  $G = (A \cup B, E)$ . To show that  $G$  is elementary, we show that  $A$  and  $B$  are the only optimal vertex covers of  $G$ . We assume that the edges of the perfect matching in  $G$  are  $\{a_1b_1, a_2b_2, \dots, a_nb_n\}$  and that the special subset  $S$  is induced by the vertices based on the edges  $\{a_1b_1, \dots, a_kb_k\}$ . Let  $\alpha$  and  $\beta$  denote the endpoints of the edge in  $G_S$  created by the contraction of  $S$ . Finally, let  $(A', B')$  denote the partition of  $G_S$ .

Note that  $G[S]$  is assumed to be elementary. If  $X$  is an optimal vertex cover for  $G$ , then  $X \cap S$  is a vertex cover for  $G[S]$  of size  $k$ , since  $G \setminus S$  has a matching of size  $(n - k)$ . Observe that:

- either  $X \cap A = \{a_1, \dots, a_k\}$  and  $X \cap B = \emptyset$ , or
- $X \cap A = \emptyset$  and  $X \cap B = \{b_1, \dots, b_k\}$ ,

because any other subset of  $k$  vertices that forms a valid vertex cover for  $G[S]$  would contradict the assumption that  $G[S]$  is elementary.

Now assume that  $G$  has a vertex cover  $X$  such that  $X \neq A$  and  $X \neq B$ . Without loss of generality, assume that  $X \cap A = \{a_1, \dots, a_k\}$  (the argument for the other case is symmetric). We let  $X_A := X \cap \{a_{k+1}, \dots, a_n\}$  and  $X_B = X \cap \{b_{k+1}, \dots, b_n\}$ . Note that  $X_B \neq \emptyset$  because  $X \neq A$ .

We now claim that  $X' := X_A \cup X_B \cup \{\alpha\}$  is an optimal vertex cover in  $G_S$  that is different from both  $A'$  and  $B'$ . It is clear that  $|X'| = (n - k + 1)$ , thus the size of  $X'$  is  $\text{mvc}(G_S)$ . Further, since  $\alpha \in A'$  and  $X_B \neq \emptyset$ ,  $X' \neq A'$  and  $X' \neq B'$ . It remains to be shown that  $X'$  covers all edges in  $G_S$ .

Note that if any edge  $a_ib_j$  in  $G_S[\{a_{k+1}, \dots, a_n\} \cup \{b_{k+1}, \dots, b_n\}]$  is not covered by  $X'$ , then the  $a_ib_j$  is also not covered by  $X$  in  $G$ . All edges incident on  $\alpha$  are also covered. Now suppose an edge of the form  $\beta a_\ell$  is not covered by  $X'$  for some  $\ell \in \{a_{k+1}, \dots, a_n\}$ . Note that for this edge to be present in  $G_S$ , by the definition of the contraction operation, there must have been an edge of the form  $b_ia_\ell$ , for some  $i \in [k]$ . Note that if  $\beta a_\ell$  is not covered by  $X'$  then  $b_ia_\ell$  is not covered by  $X$  in  $G$  either, which contradicts our assumption that  $X$  was a vertex cover in  $G$ .

This shows that  $G_S$  has an optimal vertex cover different from both  $A'$  and  $B'$ , but this contradicts our assumption that  $G_S$  was elementary to begin with. Therefore it must be the case that  $G$  is also elementary. ◀

► **Definition 16** (Alternating cycle). *Let  $G$  be a connected bipartite graph with no degree 1 vertex and a perfect matching  $M = \{a_1b_1, a_2b_2, \dots, a_nb_n\}$ . We define a cycle  $a_{i_1}b_{i_1}a_{i_2}b_{i_2} \dots b_{i_k}a_{i_k}a_{i_1}$  in  $G$  as an **alternating cycle** where  $i_1, i_2, \dots, i_k \in [n]$  and  $2 \leq k \leq n$ .*

► **Proposition 17** (No alternating cycle). *Let  $G$  be a maximal contraction graph. Then  $G$  cannot contain any alternating cycle.*

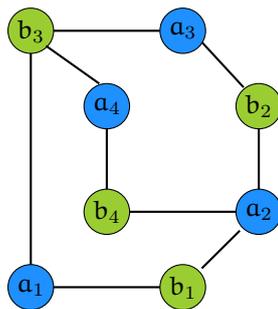
**Proof.** Let  $G$  be a maximal contraction graph. It is clear that  $G$  is a connected bipartite graph with a perfect matching. Let  $M = \{a_1b_1, a_2b_2, \dots, a_nb_n\}$  be a perfect matching of  $G$ . If  $n = 1$ ,  $G$  cannot contain any cycle and hence cannot contain any alternating cycle. Let  $n \geq 2$  and let  $C = a_{i_1}b_{i_1}a_{i_2}b_{i_2} \dots b_{i_k}a_{i_k}a_{i_1}$  be an alternating cycle in  $G$ . Let  $S = V(C)$ . Clearly  $S$  is a special subset of  $V(G)$  and by Proposition 7 and Proposition 10,  $S$  is a special elementary subset of  $V(G)$ . This contradicts the fact that  $G$  is a maximal contraction graph. Thus  $G$  cannot have an alternating cycle. ◀

► **Proposition 18** (Structure of the Maximal Contraction). *Let  $G$  be a maximal contraction graph. Then  $G$  must have a degree 1 vertex.*

## 68:12 Spartan Bipartite Graphs Are Essentially Elementary

**Proof.** Let  $G$  be a maximal contraction graph. It is clear that  $G$  is a connected bipartite graph with a perfect matching. Let  $M = \{a_1b_1, a_2b_2, \dots, a_nb_n\}$  be a perfect matching of  $G$ . If  $n = 1$ , then  $G$  is a single edge and thus  $G$  has a degree 1 vertex. Now let  $n \geq 2$  and assume that  $G$  has no degree 1 vertices. Consider the set  $S = \{a_1, b_1\}$ . Since  $b_1$  is not a degree 1 vertex, we are done. So there exists  $a_{i_1} \neq a_1$  which is a neighbor of  $b_1$ . Add  $a_{i_1}$  and  $b_{i_1}$  to  $S$ . Now for each  $b_{i_p}$  for  $p = 1, 2, \dots$ , there exists  $a_{p+1}$  which is a neighbor of  $b_{i_p}$  because  $G$  has no degree 1 vertex or alternating cycle. Add  $a_{p+1}$  and  $b_{p+1}$  to  $S$ . Thus  $S$  will be an infinite set. But  $S \subset V(G)$  which is finite. Hence we have a contradiction. Thus  $G$  must have a degree 1 vertex. ◀

To sum up the proof of Lemma 3: consider any connected bipartite graph  $G$ . First we have shown that if  $G$  is Spartan, it must have a perfect matching and no degree one vertices. Therefore, a maximal contraction graph derived from  $G$  must be an edge. But note that an edge is elementary, and running the contractions in reverse to recover  $G$  preserves the property of the graph being elementary, and we have the desired conclusion. We remark that our proof shows that every connected bipartite graph with a Hamiltonian cycle is Spartan. However, it turns out that the converse is not true in the sense that there exist connected bipartite graphs that are Spartan but do not have Hamiltonian cycles: for instance, Figure 7 presents an example of a connected bipartite Spartan graph that is not Hamiltonian.



■ **Figure 7** An example of an elementary bipartite graph where the two independent sets are  $A = \{a_1, a_2, a_3, a_4\}$  and  $B = \{b_1, b_2, b_3, b_4\}$  and all the edges belong to one of the three perfect matchings:  $M_1 = \{a_1b_1, a_2b_2, a_3b_3, a_4b_4\}$ ,  $M_2 = \{a_1b_3, a_3b_2, a_2b_1, a_4b_4\}$  or  $M_3 = \{a_1b_1, a_2b_4, a_4b_3, a_3b_2\}$ . It can be verified that this graph does not have a Hamiltonian cycle.

### 3 EVC with Extra Steps

Almost all of the proofs in the previous sections rely crucially on the fact that a guard can only move one step after each attack. This gives rise to the question that what happens if multiple moves are allowed? If retracing of steps is allowed, then any guard can clean up an attack and come back to their original position. Thus this problem will be the same as the vertex cover problem and hence it is not so interesting.

We define a variant of the ETERNAL VERTEX COVER problem that we call *New Eternal Vertex Cover*. Again we have a two player game with one player as “the attacker” and the other player as “the defender”. Just like the ETERNAL VERTEX COVER problem, the defender initially places the guards on some of the vertices of the graph (of his choice). The attacker attacks an edge. In response to the attack, the defender can move each guard for at most *two* steps without retracing. The constraint is however that the defense of the attacked edge must happen in the first move itself, i.e., after the attack, at least one guard who was present on one of the endpoints of the graph must move across the attacked edge. If such a movement is not possible after a finite sequence of attacks, the attacker wins; otherwise, if

the defender has a strategy to defend the graph  $G$  against an infinite sequence of attacks, the defender wins. There can only be one guard per vertex in the configuration before the attack and the configuration after the attack is defended. However, more than one guard can cross a vertex during the reconfiguration.

We define the *New Eternal Vertex Cover Number* of a graph  $G$  as the smallest number of guards required in this new setting such that the defender has a winning strategy. We denote this number by  $\text{nevc}(G)$ . It is clear that since the defense must happen in the first step of a guard, if the vertices occupied by guards do not form a vertex cover, the attacker wins. Therefore, we have  $\text{mvc}(G) \leq \text{nevc}(G)$  for any graph  $G$ . Also, since  $\text{evc}(G)$  many guards can defend an infinite sequence of attacks on a graph  $G$  with each guard moving just one step after each attack, it can be seen that for any graph  $G$ , we have  $\text{nevc}(G) \leq \text{evc}(G)$ .

► **Remark.** These two inequalities imply that for any Spartan graph  $G$ , we have  $\text{mvc}(G) = \text{nevc}(G) = \text{evc}(G)$ . However, there are graph classes where these inequalities are not strict.

We show that computing the *New Eternal Vertex Cover* number for a given graph  $G$  is NP-hard in the next lemma. We use a reduction from the *Vertex Cover* problem. Given an input instance  $(G, k)$  (where  $|V(G)| = n$  and  $k < n - 1$ ) of the *Vertex Cover* problem, we construct an equivalent instance of the *New Eternal Vertex Cover* problem by adding a global vertex  $\star$  to the graph  $G$ , i.e., make the new vertex  $\star$  adjacent to every other vertex in  $G$ . It turns out that  $G$  has a vertex cover of size  $k$  if and only if the defender has a winning strategy in the new setting using  $k + 1$  guards. The proof of equivalence is deferred to the full version due to lack of space.

► **Lemma 19** ( $\star$ ). *The New Eternal Vertex Cover problem is NP-hard.*

► **Lemma 20.** *For any connected graph  $G$ , the defender has a winning strategy using  $\text{mvc}(G) + 1$  guards, i.e.,  $\text{nevc}(G) \leq \text{mvc}(G) + 1$  for any graph  $G$ .*

**Proof.** For any graph  $G$ , we give a winning strategy for the defender in the new setting using  $\text{mvc}(G) + 1$  guards.

Let  $\text{mvc}(G) = k$  and let  $C_1, C_2, \dots, C_q$  be the connected components of the subgraph induced by the vertices of a  $k$ -sized vertex cover of  $G$ . Place one guard on each of these vertices of this vertex cover (say  $S$ ) and one guard in the independent set (say on the vertex  $w$ ).

Now suppose some edge is attacked such that both its endpoints have guards, then the guards can exchange places and the configuration is restored.

Suppose some edge  $uv$  is attacked such that  $u \in S$  and  $v \in V(G) \setminus S$ . Then the guard on  $u$  moves to  $v$ . Suppose  $w$  is adjacent to some vertex which lies in the same connected component of  $S$  as  $w$ , then the guards can each move one step along a path from  $w$  to  $u$  such that there is a guard on  $u$  and all the vertices of  $S$  at the end of this movement and now no guard on  $w$ . But we still have all the vertices of  $S$  with one guard each and one guard in the independent set (now on  $v$ ). Thus we have a configuration just like the initial configuration and each strategy can be implemented here accordingly.

Suppose  $u$  and any neighbour of  $w$  do not lie in the same connected component of  $S$ . As  $G$  is connected, there is a path from  $w$  to  $u$  in  $G$  and this path contains some vertices of  $V(G) \setminus S$ . Let this path be  $w = u_1 u_2 \dots u_\ell = u$ . We trigger a movement of guards along this path as follows: If  $u_{\ell-1}$  has a guard, this guard moves to  $u$ , otherwise  $u_{\ell-2}$  must have a guard because  $S$  is a vertex cover. This guard moves to  $u_{\ell-1}$  and then to  $u$ . Similarly for each  $u_i \in S$  where  $i \in \{\ell - 2, \ell - 3, \dots, 2\}$ , if  $u_{i-1}$  has a guard, the guard on  $u_{i-1}$  moves to  $u_i$ . Otherwise  $u_{i-2}$  must have a guard (because  $S$  is a vertex cover) which moves to  $u_{i-1}$

and then to  $u_i$ . It can be observed that at the end of this movement, all vertices in  $S$  have a guard and  $v$  has a guard and  $w$  does not have a guard. Therefore, we have the same situation as before where all the vertices in the vertex cover have a guard and one guard outside the vertex cover. Thus we have shown that  $\text{mvc}(G) + 1$  many guards are sufficient to defend  $G$  against any sequence of attacks in this new setting.  $\blacktriangleleft$

► **Lemma 21.** *If  $G$  is any graph without a degree-1 vertex, then  $\text{nevc}(G) = \text{mvc}(G)$ .*

**Proof.** It is sufficient to show that  $\text{nevc}(G) \leq \text{mvc}(G)$ , i.e., the defender has a winning strategy using  $\text{mvc}(G)$  many guards. Also it is sufficient to only look at connected graphs.

Let  $\text{mvc}(G) = k$  and let  $S$  be a minimum sized vertex cover of  $G$ . The defender places one guard on each vertex of  $S$ . If any edge with both endpoints in  $S$  is attacked, then the guards exchange their positions and we are back to the same configuration.

Suppose any edge  $uv$  such that  $u \in S$  and  $v \notin S$  is attacked. The guard on  $u$  moves to  $v$ . Since  $v$  is not a degree 1 vertex, it must have a neighbour  $w$  other than  $u$ . Depending on whether  $w$  and  $u$  lie in the same connected component in  $G[S]$  or not, we can trigger a movement of guards just like the proof of Lemma 20 such that each guard moves at most two steps and there is no guard on  $v$  at the end of this movement and there is a guard on  $u$ . Thus the vertex cover  $S$  is restored and we have a winning strategy using  $\text{mvc}(G)$  many guards.  $\blacktriangleleft$

► **Lemma 22.** *For any graph  $G$ ,  $\text{nevc}(G) = \text{mvc}(G)$  if and only if for each degree-1 vertex  $v$  of  $G$ , there exists a minimum sized vertex cover  $S_v$  of  $G$  which contains  $v$ .*

**Proof.** Let  $G$  be a graph such that  $\text{mvc}(G) = k$  and there exists a degree-1 vertex  $v$  such that no  $k$ -sized vertex cover of  $G$  contains  $v$ . Let  $u$  be the neighbour of  $v$ , then any minimum sized vertex cover of  $G$  must contain  $u$ . Therefore, in the initial configuration there must be a guard on  $u$  and no guard on  $v$ . If the attacker attacks the edge  $uv$ , then the defender must move the guard on  $u$  to  $v$ . Since  $v$  has no other neighbour than  $u$  and retracing is not allowed, the guard on  $v$  cannot move anywhere else. Now since there is no vertex cover of size  $k$  which contains  $v$ , the guards cannot reconfigure to form a vertex cover, no matter how the other guards arrange themselves. Thus  $\text{mvc}(G)$  many guards are not sufficient to protect  $G$  and thus  $\text{nevc}(G) \neq \text{mvc}(G)$ .

Now suppose that for every degree-1 vertex  $v$  of  $G$ , there exists a minimum sized vertex cover  $S_v$  of  $G$  which contains  $v$ . Let the size of the minimum sized vertex cover of  $G$  be  $k$ . We now describe a strategy to defend an attack on  $G$  with  $k$  guards. Place each guard on a vertex of a minimum sized vertex cover (say  $S$ ) of  $G$ . Without loss of generality, we assume that some edge  $uv$  such that  $u \in S$  and  $v \notin S$  is attacked. The guard on  $u$  is forced to move to  $v$ . If  $v$  is a vertex with degree 2 or more, then we mimic the strategy in Lemma 20 to get all the guards back on  $S$ . If  $v$  is a degree-1 vertex, we show that it is possible to transfer the guards from  $S$  to  $S_v$  (where  $S_v$  is the minimum sized vertex cover containing  $v$ ).

Denote  $S \cap S_v$  by  $P$  and  $(V(G) \setminus S \cap S_v) \setminus \{v\}$  by  $Q$ . Here  $P$  are the vertices which need to retain a guard and  $Q$  are the vertices which need to gain a guard in order to reconfigure from  $S$  to  $S_v$ . Let  $|P| = p$  and  $|Q| = q$ . Clearly  $p + q + 1 = k$ , i.e.,  $k - p = q + 1$ . Therefore  $|S \setminus P| = |Q \cup \{v\}|$ . We show that there is a perfect matching between these two sets. Notice that there cannot be an edge with both endpoints in  $S \setminus P$  because these vertices are not in  $S_v$  and  $S_v$  is a vertex cover. Suppose there is no perfect matching between  $S \setminus P$  and  $Q \cup \{v\}$ , then there exists a set  $A \in S \setminus P$  such that  $|N(A) \cap (Q \cup \{v\})| < |A|$ . But this is not possible as  $(S \setminus P) \setminus A \cup N(A)$  will be a vertex cover of  $G$  of size less than  $k$ . Thus there exists a perfect matching between  $S \setminus P$  and  $Q \cup \{v\}$  which can be used to reconfigure the guards from  $S$  to  $S_v$ .  $\blacktriangleleft$

► **Corollary 23** ( $\star$ ). *There exists a polynomial time algorithm for finding the New Eternal Vertex Cover number on bipartite graphs.*

Note that if we allow the guards to move for an arbitrary number of steps without retracing after each attack, Lemma 22 still holds, i.e., there are families of graphs (for instance, star graphs) such that even allowing the guards to move for an arbitrary number of steps does not make them “Spartan” in the new setting. Thus the power of one extra step subsumes the power of any number of additional steps.

## 4 Concluding Remarks

We showed that a natural sufficient condition for when a graph is Spartan (i.e, when  $\text{evc}(G) = \text{mvc}(G)$ ) is also necessary in the context of bipartite graphs. Motivated by our proof we extend the notion of eternal vertex cover to a variant where the guards are allowed to move more than one step on their turn, and completely characterize the number of guards needed in terms of  $\text{mvc}$ : indeed, we show that one extra guard suffices, and is needed only if the graph has a degree one vertex that is not contained in any minimum vertex cover.

It would be interesting to see what happens if the defense can happen in the second step of a guard. We also showed that while the new variant remains computationally hard, unlike the original problem, it is in fact solvable in polynomial time on bipartite graphs. Generalizing our structural results beyond bipartite graphs is an interesting direction for future work.

---

## 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.
- 2 Jasine Babu, L. Sunil Chandran, Mathew C. Francis, Veena Prabhakaran, Deepak Rajendraprasad, and J. Nandini Warriar. On graphs with minimal eternal vertex cover number. In Sudebkumar Prasant Pal and Ambat Vijayakumar, editors, *Proceedings of the 5th International Conference on Algorithms and Discrete Applied Mathematics, CALDAM*, volume 11394 of *Lecture Notes in Computer Science*, pages 263–273. Springer, 2019.
- 3 Jasine Babu, L. Sunil Chandran, Mathew C. Francis, Veena Prabhakaran, Deepak Rajendraprasad, and Nandini J Warriar. On graphs whose eternal vertex cover number and vertex cover number coincide. *Discrete Applied Mathematics (in press)*, 2021.
- 4 Reinhard Diestel. *Graph theory*. Springer, 2017.
- 5 Fedor V. Fomin, Serge Gaspers, Petr A. Golovach, Dieter Kratsch, and Saket Saurabh. Parameterized algorithm for eternal vertex cover. *Information Processing Letters*, 110(16):702–706, 2010.
- 6 Gábor Hetyei. Rectangular configurations which can be covered by  $2 \times 1$  rectangles. *Pécsi Tan. Foisk. Közl.*, 8:351–367, 1964.
- 7 William Klostermeyer and Christina M. Mynhardt. Graphs with equal eternal vertex cover and eternal domination numbers. *Discrete Mathematics*, 311(14):1371–1379, 2011.
- 8 William F. Klostermeyer and Christina M. Mynhardt. Edge protection in graphs. *Australas. J Comb.*, 45:235–250, 2009.
- 9 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.



# On the Finite Variable-Occurrence Fragment of the Calculus of Relations with Bounded Dot-Dagger Alternation

Yoshiki Nakamura  

Tokyo Institute of Technology, Japan

---

## Abstract

We introduce the *k*-variable-occurrence fragment, which is the set of terms having at most *k* occurrences of variables. We give a sufficient condition for the decidability of the equational theory of the *k*-variable-occurrence fragment using the finiteness of a monoid. As a case study, we prove that for Tarski's calculus of relations with bounded dot-dagger alternation (an analogy of quantifier alternation in first-order logic), the equational theory of the *k*-variable-occurrence fragment is decidable for each *k*.

**2012 ACM Subject Classification** Theory of computation → Equational logic and rewriting

**Keywords and phrases** Relation algebra, First-order logic, Decidable fragment, Monoid

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.69

**Related Version** *Extended Version*: <https://arxiv.org/abs/2307.05046> [23]

**Supplementary Material** *Dataset*: [https://bitbucket.org/yoshikinakamura/k-vo\\_cor\\_with\\_bounded\\_dot\\_dagger](https://bitbucket.org/yoshikinakamura/k-vo_cor_with_bounded_dot_dagger) [24]; archived at `swh:1:dir:98b3500e356c459f3d3db7e5c2ccca4863a5359c`

**Funding** *Yoshiki Nakamura*: This work was supported by JSPS KAKENHI Grant Number JP21K13828.

**Acknowledgements** We would like to thank the anonymous reviewers for their useful comments.

## 1 Introduction

Since *the satisfiability problem of first-order logic* is undecidable [8, 30] in general, (un-)decidable classes of first-order logic are widely studied [7]; for example, the undecidability holds even for the *Kahr–Moore–Wang (KMW) class*<sup>1</sup>  $[\forall\exists\forall, (0, \omega)]$  [14], but it is decidable for the *Bernays–Schönfinkel–Ramsey (BSR) class*  $[\exists^*\forall^*, \text{all}] = [3, 27]$ . The calculus of relations (CoR) [28], revived by Tarski, is an algebraic system on binary relations; its expressive power is equivalent to that of the three-variable fragment of first-order logic with equality [28, 29], w.r.t. binary relations. The equational theory of CoR is undecidable [28, 29]<sup>2</sup> in general, which follows from the undecidability of the KMW class, but, for example, it is decidable for the *(existential) positive fragment* [2, 26] and the *existential fragment* [22] of CoR, which follows from the decidability of the BSR class. On the undecidability of CoR, the undecidability holds even for the 1-variable fragment [16] and even for the 1-variable fragment only with union, composition, and complement [19], where the *k*-variable fragment denotes the set of terms having at most *k* variables.

---

<sup>1</sup> Recall the notation for *prefix-vocabulary classes* [7, Def. 1.3.1]. E.g.,  $[\forall\exists\forall, (0, \omega)]$  denotes the set of prenex sentences  $\varphi$  of first-order logic without equality, function symbols, nor constants such that the quantifier prefix of  $\varphi$  is  $\forall\exists\forall$ ;  $\varphi$  has  $\omega$  (countably infinitely many) binary relation symbols and  $\varphi$  does not have 1- nor *i*-ary relation symbols for  $i \geq 3$ .

<sup>2</sup> In [29], the undecidability of the equational theory is shown for more general classes of relation algebras.



© Yoshiki Nakamura;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 69; pp. 69:1–69:15

Leibniz International Proceedings in Informatics



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

Then, from the undecidability result for the 1-variable fragment of CoR [16, 19] above, the following natural question arises – Is it decidable for the  $k$ -variable-occurrence fragment of CoR? Here, the  $k$ -variable-occurrence fragment denotes the set of terms having at most  $k$  occurrences of variables. For example, when  $a, b$  are variables and  $l$  is a constant, the term  $(a \cdot b) \cdot (l \cdot (a \cdot b))$  has 4 occurrences of variables and 1 occurrence of constants; thus, this term is in the 4-variable-occurrence fragment (cf. the term is in the 2-variable fragment since the variables  $a, b$  occur). While one may seem that this restriction immediately implies the decidability, the equational theory of the  $k$ -variable-occurrence fragment on some (single) algebra is undecidable in general even when  $k = 0$  (Remark 2).

Our contribution is to prove that the equational theory of the  $k$ -variable-occurrence fragment is decidable for CoR *with bounded dot-dagger alternation*, where the dot-dagger alternation [20, 21] is an analogy of the quantifier alternation in first-order logic. Note that the equational theory of the  $k$ -variable fragment is undecidable in general for CoR [16, 19] (even with bounded dot-dagger alternation (Prop. 24)).

Our strategy is to prove that the number of terms in the  $k$ -variable-occurrence fragment is finite up to the semantic equivalence relation. To this end, (1) we decompose terms as much as possible; and then (2) we show that each decomposed part is finite up to the semantic equivalence relation by collecting valid equations. By the preprocessing of (1), one can see that for (2), essentially, it suffices to prove the finiteness of some monoid (using the method of Sect. 2). Its finiteness is not clear, as it is undecidable whether a (finitely presented) monoid is finite in general; but, fortunately, we can prove the finiteness (Thm. 25) by finding valid equations (Fig. 1).

The rest of this paper is structured as follows. Sect. 2 introduces the  $k$ -variable-occurrence fragment for general algebras and gives a framework to prove the decidability from the finiteness of a monoid. Sect. 3 recalls the syntax and semantics of CoR and the dot-dagger alternation hierarchy. In Sect. 4, based on Sect. 2, we prove that the equational theory of CoR with bounded dot-dagger alternation is decidable. Sect. 5 concludes this paper.

We write  $\mathbb{N}$  for the set of all non-negative integers. For  $l, r \in \mathbb{N}$ , we write  $[l, r]$  for the set  $\{i \in \mathbb{N} \mid l \leq i \leq r\}$ . For a set  $A$ , we write  $\#A$  for the cardinality of  $A$  and  $\wp(A)$  for the power set of  $A$ . For a set  $A$  and an equivalence relation  $\sim$  on  $A$ , we write  $A/\sim$  for the quotient set of  $A$  by  $\sim$  and  $[a]_{\sim}$  for the equivalence class of an element  $a \in A$  on  $\sim$ .

## 2 On the $k$ -variable-occurrence fragment

We fix  $\Sigma$  as a non-empty *finite* set of variables. We fix  $S$  as a *finite* algebraic signature;  $S$  is a map from a finite domain (of functions) to  $\mathbb{N}$ . For each  $\langle f, n \rangle \in S$ , we write  $f^{(n)}$ ; it is the function symbol  $f$  with arity  $n$ . We also let  $S^{(n)} \triangleq \{f^{(m)} \in S \mid m = n\}$ . The set  $\mathbf{T}$  of  $S$ -terms over  $\Sigma$  is defined as the minimal set closed under the following two rules:  $a \in \Sigma \implies a \in \mathbf{T}$ ;  $(f^{(n)} \in S \text{ and } t_1, \dots, t_n \in \mathbf{T}) \implies f(t_1, \dots, t_n) \in \mathbf{T}$ .

An  $S$ -algebra  $A$  is a tuple  $\langle |A|, \{f^A\}_{f^{(n)} \in S} \rangle$ , where  $|A|$  is a non-empty finite set and  $f^A: |A|^n \rightarrow |A|$  is an  $n$ -ary map for each  $f^{(n)} \in S$ . A *valuation*  $\mathbf{v}: \Sigma \rightarrow |A|$  on an  $S$ -algebra  $A$  is a map; we write  $\hat{\mathbf{v}}: \mathbf{T} \rightarrow |A|$  for the unique homomorphism extending  $\mathbf{v}$ . For a class  $\mathcal{C}$  of  $S$ -algebras, the equivalence relation  $\sim_{\mathcal{C}}$  on  $\mathbf{T}$  is defined by:  $t \sim_{\mathcal{C}} s \iff \hat{\mathbf{v}}(t) = \hat{\mathbf{v}}(s)$  for all valuations  $\mathbf{v}$  on all algebras in  $\mathcal{C}$ . For a set  $T \subseteq \mathbf{T}$ , the equational theory of  $T$  over  $\mathcal{C}$  is the set  $\{\langle t, s \rangle \in T^2 \mid t \sim_{\mathcal{C}} s\}$ .

► **Definition 1** (*k*-variable-occurrence fragment). For an *S*-term  $t$ , let  $\text{vo}(t)$  be the number of occurrences of variables in  $t$ :  $\text{vo}(t) \triangleq \begin{cases} 1 & (t \in \Sigma) \\ \sum_{i=1}^n \text{vo}(t_i) & (t = f(t_1, \dots, t_n)) \end{cases}$ . For each set  $T$  of *S*-terms, the *k*-variable-occurrence fragment  $T_{(\text{vo} \leq k)}$  is the set  $\{t \in T \mid \text{vo}(t) \leq k\}$ . (Similarly, let  $T_{(\text{vo} = k)} \triangleq \{t \in T \mid \text{vo}(t) = k\}$ .) Clearly,  $T = \bigcup_{k \in \mathbb{N}} T_{(\text{vo} \leq k)}$ .

► **Remark 2.** The equational theory of the *k*-variable-occurrence fragment is undecidable in general, even when  $k = 0$ . It follows from the reduction from the word problem for monoids. Let  $M = \langle |M|, \circ^M, \mathbf{1}^M \rangle$  be a (finitely) presented monoid with finite generators  $C = \{c_1, \dots, c_l\}$  such that the word problem for  $M$  is undecidable (by Markov [17] and Post [25]). We define  $S \triangleq \{c_1^{(1)}, \dots, c_l^{(1)}\} \cup \{\mathbf{1}^{(0)}\}$  and the *S*-algebra  $A \triangleq \langle |A|, \{f^A\}_{f^{(n)} \in S} \rangle$  by:  $|A| = |M|$ ;  $c_i^A(x) = c_i \circ^M x$  for  $i \in [1, l]$ ;  $\mathbf{1}^A = \mathbf{1}^M$ . By definition, for all two words  $a_1 \dots a_n, b_1 \dots b_m$  over  $C$ : they are equivalent in  $M$  iff  $a_1(a_2(\dots a_n(\mathbf{1}) \dots)) \sim_{\{A\}} b_1(b_2(\dots b_m(\mathbf{1}) \dots))$ .

In the rest of this section, we fix  $\mathcal{C}$  as a class of *S*-algebras.

## 2.1 On the finiteness of *k*-variable-occurrence fragment: from 1 to *k*

How can we show the decidability of the equational theory of the *k*-variable-occurrence fragment? We consider proving it from the finiteness up to the semantic equivalence relation:

► **Proposition 3** (Cor. of [5, 15] for the complexity). Let  $T \subseteq \mathbf{T}$  be a subterm-closed<sup>3</sup> set. If the set  $T/\sim_{\mathcal{C}}$  is finite, the equational theory of  $T$  over  $\mathcal{C}$  is decidable. Moreover, it is decidable in DLOGTIME-uniform NC<sup>1</sup> if the input is given as a well-bracketed string.

**Proof Sketch.** Because  $\mathcal{C}$  is fixed and  $T/\sim_{\mathcal{C}}$  is finite, for each  $t \in T$ , one can calculate the index of the equivalence class of  $t$  on  $\sim_{\mathcal{C}}$  by using the (finite and possibly partial) Cayley table of each operator; thus, the equational theory is decidable. Moreover, according to this algorithm, if the input is given as a well-bracketed string, one can also construct a parenthesis context-free grammar such that for all  $t, s \in T$ , the well-bracketed string encoding the equation  $t = s$  is in the language iff  $t \sim_{\mathcal{C}} s$ . Hence, the complexity is shown because every language recognized by a parenthesis context-free grammar is in ALOGTIME [5, 6] (ALOGTIME is equivalent to DLOGTIME-uniform NC<sup>1</sup> [18]). ◀

For the *k*-variable-occurrence fragment, the finiteness of  $T_{(\text{vo} \leq 1)}$  (with Prop. 3) can imply the decidability of the equational theory of  $T_{(\text{vo} \leq k)}$  (Lem. 6) by the following decomposition lemma. Here, we write  $t[s/a]$  for the term  $t$  in which each  $a$  has been replaced with  $s$ .

► **Lemma 4.** Let  $T \subseteq \mathbf{T}$  be a subterm-closed set. Let  $k \geq 2$ ,  $a \in \Sigma$ . Then, for all  $t \in T_{(\text{vo} = k)}$ , there are  $t_0 \in T_{(\text{vo} \leq 1)}$ ,  $f^{(n)} \in S$ ,  $t_1, \dots, t_n \in T_{(\text{vo} \leq k-1)}$  such that  $t = t_0[f(t_1, \dots, t_n)/a]$ .

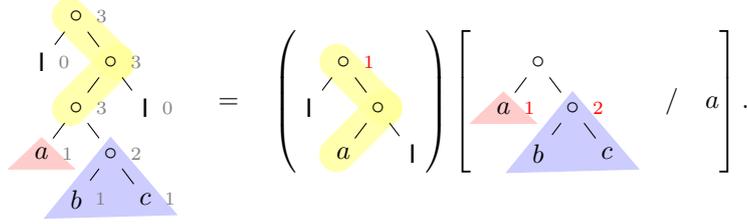
**Proof.** By induction on  $t$ . Since  $k \geq 2$ , there are  $g^{(m)} \in S$ ,  $s_1, \dots, s_m \in T$  s.t.  $t = g(s_1, \dots, s_m)$  and  $\sum_{i=1}^m \text{vo}(s_i) = k$ . Case  $\text{vo}(s_i) \leq k-1$  for all  $i$ : By letting  $t_0 \triangleq a$ , we have  $t = t_0[g(s_1, \dots, s_m)/a]$ . Otherwise: Let  $i$  be s.t.  $\text{vo}(s_i) = k$ . Since  $s_i \in T_{(\text{vo} = k)}$ , let  $u \in T_{(\text{vo} \leq 1)}$ ,  $f^{(n)} \in S$ ,  $t_1, \dots, t_n \in T_{(\text{vo} \leq k-1)}$  be the ones obtained by IH w.r.t.  $s_i$ , so that  $s_i = u[f(t_1, \dots, t_n)/a]$ . By letting  $t_0 \triangleq g(s_1, \dots, s_{i-1}, u, s_{i+1}, \dots, s_m)$ , we have:

<sup>3</sup> A set  $T \subseteq \mathbf{T}$  is subterm-closed if for every  $t \in T$ , if  $s$  a subterm of  $t$ , then  $s \in T$ .

$$\begin{aligned}
 t_0[f(t_1, \dots, t_n)/a] &= g(s_1, \dots, s_{i-1}, u, s_{i+1}, \dots, s_m)[f(t_1, \dots, t_n)/a] \\
 &= g(s_1, \dots, s_{i-1}, u[f(t_1, \dots, t_n)/a], s_{i+1}, \dots, s_m) \quad (\text{vo}(s_j) = 0 \text{ if } j \neq i) \\
 &= g(s_1, \dots, s_{i-1}, s_i, s_{i+1}, \dots, s_m) = t. \quad (s_i = u[f(t_1, \dots, t_n)/a])
 \end{aligned}$$

Hence, this completes the proof.  $\blacktriangleleft$

► **Example 5** (of Lem. 4). If  $S = \{\circ^{(2)}, \text{l}^{(0)}\}$  and  $a, b, c \in \Sigma$ , the term  $t = \text{l} \circ ((a \circ (b \circ c)) \circ \text{l}) \in \mathbf{T}_{(\text{vo} \leq 3)}$  has the following decomposition:  $t = (\text{l} \circ (a \circ \text{l})) [a \circ (b \circ c) / a]$ . Then  $\text{l} \circ (a \circ \text{l}) \in \mathbf{T}_{(\text{vo} \leq 1)}$  and  $a, (b \circ c) \in \mathbf{T}_{(\text{vo} \leq 2)}$ . The following is an illustration of the decomposition, where the number written in each subterm  $s$  denotes  $\text{vo}(s)$ :



Using this decomposition iteratively, we have the following:

► **Lemma 6.** *Let  $T \subseteq \mathbf{T}$  be a subterm-closed set. Assume that  $T_{(\text{vo} \leq 1)} / \sim c$  is finite. Then, for each  $k \in \mathbb{N}$ , the set  $T_{(\text{vo} \leq k)} / \sim c$  is finite.*

**Proof.** It suffices to prove: for all  $k \geq 2$ ,  $T_{(\text{vo}=k)} / \sim c$  is finite. By induction on  $k$ . We have:

$$\begin{aligned}
 &\# (T_{(\text{vo}=k)} / \sim c) \\
 &\leq \# (\{t_0[f(t_1, \dots, t_n)/a] \mid f^{(n)} \in S, t_1, \dots, t_n \in T_{(\text{vo} \leq k-1)}, t_0 \in T_{(\text{vo} \leq 1)}\} / \sim c) \quad (\text{Lem. 4}) \\
 &\leq \sum_{f^{(n)} \in S} \# (\{t_0[f(t_1, \dots, t_n)/a] \mid t_1, \dots, t_n \in T_{(\text{vo} \leq k-1)}, t_0 \in T_{(\text{vo} \leq 1)}\} / \sim c).
 \end{aligned}$$

Then the set  $\{t_0[f(t_1, \dots, t_n)/a] \mid t_1, \dots, t_n \in T_{(\text{vo} \leq k-1)}, t_0 \in T_{(\text{vo} \leq 1)}\} / \sim c$  is finite because  $T_{(\text{vo} \leq k-1)} / \sim c$  is finite (by IH) and  $\sim c$  satisfies the congruence law. Thus, the last term above is finite since  $S$  is finite. Hence  $T_{(\text{vo}=k)} / \sim c$  is finite.  $\blacktriangleleft$

## 2.2 The monoid of the 1-variable-occurrence fragment

Thanks to Lem. 6, we can focus on the 1-variable-occurrence fragment. For the 1-variable-occurrence fragment, it suffices to consider a monoid. For a set  $A$  of characters, we write  $A^*$  for the set of all words (i.e., finite sequences) over the alphabet  $A$ . We write  $wv$  for the concatenation of words  $w$  and  $v$  and write  $\varepsilon$  for the empty word. We write  $\|w\|$  for the length of a word  $w$ .

► **Definition 7.** *Let  $\dot{\Sigma}$  be the (possibly infinite) set of characters defined by:*

$$\dot{\Sigma} \triangleq \bigcup_{f^{(n)} \in S, i \in [1, n]} \{f(t_1, \dots, t_{i-1}, \_, t_{i+1}, \dots, t_n) \mid \forall j \in [1, n] \setminus \{i\}, t_j \in \mathbf{T}_{(\text{vo} \leq 0)}\}.$$

( $\_$  denotes “blank”.) For a word  $w \in \dot{\Sigma}^*$  and a term  $t \in \mathbf{T}$ , let  $w[t]$  be the term defined by:

$$w[t] \triangleq \begin{cases} f(t_1, \dots, t_{i-1}, w'[t], t_{i+1}, \dots, t_n) & (w = f(t_1, \dots, t_{i-1}, \_, t_{i+1}, \dots, t_n)w') \\ t & (w = \varepsilon) \end{cases}.$$

► **Example 8** (of Def. 7). If  $S = \{\circ^{(2)}, \mathbf{l}^{(0)}\}$ , then  $\dot{\Sigma} = \{(\mathbf{l} \circ \_), ((\mathbf{l} \circ \mathbf{l}) \circ \_), \dots (\_ \circ \mathbf{l}), \dots\}$ . For example, if  $w = ((\mathbf{l} \circ \mathbf{l}) \circ \_)(\_ \circ \mathbf{l})(\mathbf{l} \circ \_)$ , we have:

$$\begin{aligned} w[a] &= (((\mathbf{l} \circ \mathbf{l}) \circ \_)(\_ \circ \mathbf{l})(\mathbf{l} \circ \_)[a]) = (\mathbf{l} \circ \mathbf{l}) \circ ((\_ \circ \mathbf{l})(\mathbf{l} \circ \_)[a]) \\ &= (\mathbf{l} \circ \mathbf{l}) \circ (((\mathbf{l} \circ \_)[a]) \circ \mathbf{l}) \\ &= (\mathbf{l} \circ \mathbf{l}) \circ ((\mathbf{l} \circ \varepsilon[a]) \circ \mathbf{l}) = (\mathbf{l} \circ \mathbf{l}) \circ ((\mathbf{l} \circ a) \circ \mathbf{l}). \end{aligned}$$

► **Proposition 9.** For all  $t \in \mathbf{T}_{(\mathbf{v}_0 \leq 1)}$ , there are  $w \in \dot{\Sigma}^*$  and  $s \in S^{(0)} \cup \Sigma$  s.t.  $t = w[s]$ .

**Proof.** By easy induction on  $t$ . ◀

► **Definition 10.** Let  $\sim_{\mathcal{C}}$  be the equivalence relation on  $\dot{\Sigma}^*$  defined by:

$$w \sim_{\mathcal{C}} v \iff w[a] \sim_{\mathcal{C}} v[a] \text{ where } a \in \Sigma \text{ is any variable.}$$

► **Lemma 11.** If  $\dot{\Sigma}^*/\sim_{\mathcal{C}}$  is finite, then  $\mathbf{T}_{(\mathbf{v}_0 \leq 1)}/\sim_{\mathcal{C}}$  is finite.

**Proof.** By Prop. 9 (and that the set  $S^{(0)} \cup \Sigma$  is finite). ◀

Moreover, if  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}}$  is finite, it suffices to consider a finite subset of  $\dot{\Sigma}$ , as follows:

► **Lemma 12.** Assume that  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}}$  is finite. Let  $T_0 = \{t_1, \dots, t_n\} \subseteq \mathbf{T}_{(\mathbf{v}_0 \leq 0)}$  be such that  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}} = \{[t_1]_{\sim_{\mathcal{C}}}, \dots, [t_n]_{\sim_{\mathcal{C}}}\}$ . Let  $\dot{\Sigma}_0 \subseteq \dot{\Sigma}$  be the finite set defined by:

$$\dot{\Sigma}_0 \triangleq \bigcup_{f^{(n)} \in S, i \in [1, n]} \{f(t_1, \dots, t_{i-1}, \_, t_{i+1}, \dots, t_n) \mid \forall j \in [1, n] \setminus \{i\}, t_j \in T_0\}.$$

Then  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite  $\implies \dot{\Sigma}^*/\sim_{\mathcal{C}}$  is finite.

**Proof.** For every  $a \in \dot{\Sigma}$ , there is  $b \in \dot{\Sigma}_0$  s.t.  $a \sim_{\mathcal{C}} b$ . By the congruence law of  $\sim_{\mathcal{C}}$ , for every  $w \in \dot{\Sigma}^*$ , there is some  $v \in \dot{\Sigma}_0^*$  s.t.  $w \sim_{\mathcal{C}} v$ . Since  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite, this completes the proof. ◀

► **Example 13** (of Lem. 12). If  $S = \{\circ^{(2)}, \mathbf{l}^{(0)}\}$  (so,  $\dot{\Sigma} = \{(\mathbf{l} \circ \_), ((\mathbf{l} \circ \mathbf{l}) \circ \_), \dots (\_ \circ \mathbf{l}), \dots\}$ ) and  $\mathcal{C}$  is the class of all monoids, we have:  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}} = \{[\mathbf{l}]_{\sim_{\mathcal{C}}}\}$ . Thus the set  $\dot{\Sigma}_0 = \{(\mathbf{l} \circ \_), (\_ \circ \mathbf{l})\}$  is sufficient for considering the finiteness of  $\dot{\Sigma}^*/\sim_{\mathcal{C}}$ .

Thus, to prove the finiteness of  $\mathbf{T}_{(\mathbf{v}_0 \leq k)}/\sim_{\mathcal{C}}$ , it suffices to prove that both  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}}$  and  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  are finite:

► **Lemma 14.** If  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}}$  and  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  are finite, then for each  $k \in \mathbb{N}$ , the set  $\mathbf{T}_{(\mathbf{v}_0 \leq k)}/\sim_{\mathcal{C}}$  is finite (hence, the equational theory of  $\mathbf{T}_{(\mathbf{v}_0 \leq k)}$  over  $\mathcal{C}$  is decidable).

**Proof.** We have:  $\mathbf{T}_{(\mathbf{v}_0 \leq 0)}/\sim_{\mathcal{C}}$  and  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  are finite  $\implies \dot{\Sigma}^*/\sim_{\mathcal{C}}$  is finite (by Lem. 12)  $\implies \mathbf{T}_{(\mathbf{v}_0 \leq 1)}/\sim_{\mathcal{C}}$  is finite (by Lem. 11)  $\implies \mathbf{T}_{(\mathbf{v}_0 \leq k)}/\sim_{\mathcal{C}}$  is finite (by Lem. 6). (The decidability is obtained from Prop. 3.) ◀

### 2.3 Finiteness from finding equations

For languages  $L, K$ , we write  $LK$  for the concatenation of  $L$  and  $K$ :  $LK = \{wv \mid w \in L, v \in K\}$ . For words  $w_i$  ( $i \in I$ ), we write  $\bigcup_{i \in I} w_i$  for the language  $\{w_i \mid i \in I\}$ .

For the finiteness of  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$ , we consider finding equations  $\{\langle w_i, v_i \rangle \mid i \in I\}$  and then applying the following:

► **Lemma 15.** *Let  $\dot{\Sigma}_0 \subseteq \dot{\Sigma}$  be a finite set. Let  $\langle \cdot \rangle \subseteq (\dot{\Sigma}_0^*)^2$  be a well-founded relation s.t.*

- $\langle \cdot \rangle$  satisfies the congruence law (i.e.,  $v < v' \implies wv < wv'$ );
- $\langle \cdot \rangle$  has no infinite antichains.<sup>4</sup>

Then, the following are equivalent:

1. There is a finite set  $\{\langle w_i, v_i \rangle \mid i \in I\} \subseteq \langle \cdot \rangle \cap (\sim_{\mathcal{C}})$  such that the language  $\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$  over the alphabet  $\dot{\Sigma}_0$  is cofinite.<sup>5</sup>
2.  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite.

**Proof.**  $1 \implies 2$ : By induction on the well-founded relation  $\langle \cdot \rangle$ , we prove: For every  $w \in \dot{\Sigma}_0^*$ , there is some  $v \in \dot{\Sigma}_0^* \setminus (\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*)$  such that  $w \sim_{\mathcal{C}} v$ . If  $w \in \dot{\Sigma}_0^* \setminus (\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*)$ , by letting  $v = w$ . Otherwise, since  $w \in \dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$ , there are  $i \in I$  and  $w', w'' \in \dot{\Sigma}_0^*$  such that  $w = w'v_iw''$ . By  $w'w_iw'' < w'v_iw''$  (the congruence law of  $\langle \cdot \rangle$ ) and IH, there is  $v \in \dot{\Sigma}_0^* \setminus (\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*)$  s.t.  $w'w_iw'' \sim_{\mathcal{C}} v$ . We also have  $w'v_iw'' \sim_{\mathcal{C}} w'w_iw''$  (by  $v_i \sim_{\mathcal{C}} w_i$  with the congruence law of  $\sim_{\mathcal{C}}$ ). Thus  $w'v_iw'' \sim_{\mathcal{C}} v$  (by transitivity of  $\sim_{\mathcal{C}}$ ).

$2 \implies 1$ : Let  $W \triangleq \bigcup_{X \in \dot{\Sigma}_0^*/\sim_{\mathcal{C}}} \{w \in X \mid w \text{ is minimal w.r.t. } \langle \cdot \rangle \cap X^2\}$ . Let  $\text{Subw}(W)$  be the subword closure of  $W$  (i.e., the minimal set  $W' \supseteq W$  s.t.  $w'ww'' \in W' \implies w \in W'$ ). Let  $V \triangleq (\text{Subw}(W)\dot{\Sigma}_0) \setminus \text{Subw}(W)$ . Then  $(\dot{\Sigma}_0^*V\dot{\Sigma}_0^*) = \dot{\Sigma}_0^* \setminus \text{Subw}(W)$  holds, as follows. For  $\subseteq$ : Let  $w \in \dot{\Sigma}_0^*$ ,  $v \in V$ ,  $w' \in \dot{\Sigma}_0^*$ . If we assume  $wv'w' \in \text{Subw}(W)$ , then  $v \in \text{Subw}(W)$ , but this contradicts  $v \in V$ ; thus,  $wv'w' \notin \text{Subw}(W)$ . For  $\supseteq$ : By induction on the length of  $w \in \dot{\Sigma}_0^* \setminus \text{Subw}(W)$ . If  $w \in V$ , clear. If  $w \notin V$ , let  $w = w'a$  (note that  $w \neq \varepsilon$ , because  $\varepsilon \in \text{Subw}(W)$  always by that  $W$  is not empty). Then,  $w' \in \dot{\Sigma}_0^* \setminus \text{Subw}(W)$  (if not, since  $w' \in \text{Subw}(W)$  and  $w \notin \text{Subw}(W)$ ,  $w \in V$ , reaching a contradiction). Thus by IH,  $w' \in \dot{\Sigma}_0^*V\dot{\Sigma}_0^*$ ; thus  $w \in \dot{\Sigma}_0^*V\dot{\Sigma}_0^*$ . Hence, we have  $\dot{\Sigma}_0^* \setminus (\dot{\Sigma}_0^*V\dot{\Sigma}_0^*) = \text{Subw}(W)$ . Now, the set  $W$  is finite because  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite and for each  $X \in \dot{\Sigma}_0^*/\sim_{\mathcal{C}}$ , the number of minimal elements  $w$  is finite (because  $\langle \cdot \rangle$  has no infinite antichains); thus  $\text{Subw}(W)$  is finite; thus  $V$  is finite. Let  $V = \{v_1, \dots, v_n\}$ . For every  $i \in [1, n]$ , there is  $w_i \in W \cap [v_i]_{\sim_{\mathcal{C}}}$  s.t.  $w_i < v_i$ , because  $v_i$  is not minimal w.r.t.  $\langle \cdot \rangle \cap [v_i]_{\sim_{\mathcal{C}}}^2$ . Hence,  $\{\langle w_i, v_i \rangle \mid i \in [1, n]\}$  is the desired set. ◀

The *shortlex order* (aka *length-lexicographical order*) is an example of  $\langle \cdot \rangle$  in Lem. 15 (because it is a well-ordering [4, Def. 2.2.3] and it satisfies the congruence law).

► **Example 16** (toy example of Lem. 15 ( $1 \implies 2$ )). Let  $S = \{\circ^{(2)}, \text{I}^{(0)}, -^{(1)}\}$  and  $\mathcal{C}$  be the class of groups. Let  $\dot{\Sigma}_0 = \{\text{I} \circ \_, (\_ \circ \text{I}), \_-\}$ . Then, we have the following 3 equations:

$$\varepsilon \sim_{\mathcal{C}} (\text{I} \circ \_) \qquad \varepsilon \sim_{\mathcal{C}} (\_ \circ \text{I}) \qquad \varepsilon \sim_{\mathcal{C}} (\_ -)(\_ -).$$

(This is because  $a \sim_{\mathcal{C}} (\text{I} \circ a)$ ,  $a \sim_{\mathcal{C}} (a \circ \text{I})$ , and  $a \sim_{\mathcal{C}} (a^-)^-$  hold, respectively.) Then, the language  $(\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*) = \dot{\Sigma}_0^* \setminus \{\varepsilon, \_-\}$  is cofinite. Thus, from the 3 equations, we have that  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite. (Use a shortlex order as  $\langle \cdot \rangle$  in Lem. 15.)

<sup>4</sup> This assumption is used only in the direction of  $2 \implies 1$ .

<sup>5</sup> A language  $L$  over an alphabet  $A$  is cofinite if its complemented language  $A^* \setminus L$  is finite.

While it is undecidable whether a given (finitely presented) monoid  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite [17] (see also [4, Thm. 7.3.7 with Def. 7.3.2(b)]) in general (cf. 2 of Lem. 15), it is decidable (in linear time) whether the language of a given regular expression of the form  $\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$  is cofinite (cf. 1 of Lem. 15):

► **Proposition 17.** *The following is decidable in linear time (more precisely,  $\mathcal{O}(n)$  time on a random-access machine for  $n$  the number of symbols in the given regular expression): Given a regular expression of the form  $\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$  over the alphabet  $\dot{\Sigma}_0$ , is its language cofinite?*

**Proof Sketch.** By the Aho-Corasick algorithm, we can construct a deterministic finite automaton (DFA) from a given regular expression of the form  $\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$  in linear time [1, Sect. 8]. By taking the complemented language of the DFA, it suffices to show that the following problem is in  $\mathcal{O}(n)$  time: given a DFA with  $n$  states, is its language finite? Then we can give the following algorithm: From the graph induced by the DFA, remove all the states not reachable from the starting state and remove all the states not reachable to any accepting states by using the depth-first search; check whether there exists some cycle in the graph by the depth-first search. ◀

Thus, thanks to  $1 \implies 2$  of Lem. 15, we can focus on finding a finite set of equations. While it is undecidable in general whether there exists such a set, we give a possibly non-terminating pseudo-code in Algorithm 1, which can help to find equations (e.g., Fig. 1).<sup>6</sup>

■ **Algorithm 1** Possibly non-terminating pseudo-code for ensuring the finiteness of  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$ .

---

**Require:** Given  $S, \Sigma, \sim_{\mathcal{C}}, \dot{\Sigma}_0$ . ▷  $\dot{\Sigma}_0$  is a finite alphabet of Lem. 12.

**Ensure:** Is  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  finite?

- 1:  $\Gamma \leftarrow \emptyset$  ▷ We use  $I, w_i, v_i$  to denote  $\Gamma = \{\langle w_i, v_i \rangle \mid i \in I\}$ .
- 2: **while** the language  $\dot{\Sigma}_0^*(\bigcup_{i \in I} v_i)\dot{\Sigma}_0^*$  over  $\dot{\Sigma}_0$  is not cofinite **do**
- 3:    $\langle w, v \rangle \leftarrow$  a fresh pair in  $\dot{\Sigma}_0^* \times \dot{\Sigma}_0^*$  ▷  $<$  is a binary relation in (Lem. 15).
- 4:   **if**  $w < v$  and  $w \sim_{\mathcal{C}} v$  **then**  $\Gamma \leftarrow \Gamma \cup \{\langle w, v \rangle\}$
- 5:   **end if**
- 6: **end while**
- 7: **return** True

---

► **Remark 18.** If “given  $w, v$ , does  $w \sim_{\mathcal{C}} v$  hold” is decidable, then Algorithm 1 is a semi-algorithm (i.e., if  $\dot{\Sigma}_0^*/\sim_{\mathcal{C}}$  is finite, it is terminated and returns True; otherwise, not terminated). This is because  $2 \implies 1$  of Lem. 15 also holds.

### 3 The calculus of relations with bounded dot-dagger alternation

In the remaining part of this paper, as a case study of the  $k$ -variable-occurrence fragment presented in Sect. 2, we consider the calculus of relations with bounded dot-dagger alternation. In this section, we recall the definitions of the calculus of relations (CoR) and the dot-dagger alternation hierarchy.

<sup>6</sup> Usually, to calculate  $\sim_{\mathcal{C}}$  is a bottleneck. For relaxing this problem, for example, hashing words by using some algebras in  $\mathcal{C}$  is practically useful for reducing the number of  $\sim_{\mathcal{C}}$  calls (since if the hash of two words  $w, v$  are different, then we immediately have that  $w, v$  are not equivalent w.r.t.  $\sim_{\mathcal{C}}$ ).

### 3.1 CoR: syntax and semantics

We fix  $\Sigma$  as a non-empty *finite* set of variables. Consider the finite algebraic signature  $S^{\text{CoR}} \triangleq \{\perp^{(0)}, \top^{(0)}, -^{(1)}, \cup^{(2)}, \cap^{(2)}, \text{I}^{(0)}, \text{D}^{(0)}, \cdot^{(2)}, \dagger^{(2)}\} \cup \{\pi^{(1)} \mid \pi \text{ is a map in } [1, 2]^{[1, 2]}\}$  (we consider algebras of binary relations and each  $\pi$  is used for a projection of binary relations). The set  $\mathbf{T}^{\text{CoR}}$  of CoR terms is defined as follows:

$$\begin{aligned} \mathbf{T}^{\text{CoR}} \ni t, s, u ::= & a \mid \perp \mid \top \mid s \cup u \mid s \cap u \mid s^- \\ & \mid \text{I} \mid \text{D} \mid s \cdot u \mid s \dagger u \mid s^\pi \end{aligned} \quad (a \in \Sigma, \pi \in [1, 2]^{[1, 2]}).$$

Additionally, for a term  $t$ , we use  $t^\sim$  to denote the term  $t^\sim \triangleq t^{\{1 \rightarrow 2, 2 \rightarrow 1\}}$ . Here, we use the infix notation for binary operators, the superscript notation for unary operators, and parenthesis in ambiguous situations, as usual.

For binary relations  $R, S$  on a set  $W$ , the *identity relation*  $\text{I}_W$  on  $W$ , the *difference relation*  $\text{D}_W$  on  $W$ , the *(relational) composition (relative product)*  $R \cdot S$ , the *dagger (relative sum)*  $R \dagger S$ , and the *projection*  $R^\pi$  are defined by:

$$\begin{aligned} \text{I}_W &\triangleq \{\langle x, y \rangle \in W^2 \mid x = y\} && \text{(identity)} \\ \text{D}_W &\triangleq \{\langle x, y \rangle \in W^2 \mid x \neq y\} && \text{(difference)} \\ R \cdot S &\triangleq \{\langle x, y \rangle \in W^2 \mid \exists z \in W, \langle x, z \rangle \in R \wedge \langle z, y \rangle \in S\} && \text{(relative product)} \\ R \dagger S &\triangleq \{\langle x, y \rangle \in W^2 \mid \forall z \in W, \langle x, z \rangle \in R \vee \langle z, y \rangle \in S\} && \text{(relative sum)} \\ R^\pi &\triangleq \{\langle x_1, x_2 \rangle \in W^2 \mid \langle x_{\pi(1)}, x_{\pi(2)} \rangle \in R\} && \text{(projection)}. \end{aligned}$$

A structure  $\mathfrak{A}$  is a tuple  $\langle |\mathfrak{A}|, \{a^{\mathfrak{A}}\}_{a \in \Sigma} \rangle$ , where  $|\mathfrak{A}|$  is a non-empty set and  $a^{\mathfrak{A}} \subseteq |\mathfrak{A}|^2$  is a binary relation for each  $a \in \Sigma$ . For a structure  $\mathfrak{A}$ , the *binary relation map*  $\llbracket \_ \rrbracket_{\mathfrak{A}} : \mathbf{T}^{\text{CoR}} \rightarrow \wp(|\mathfrak{A}|^2)$  is the unique homomorphism extending  $\llbracket a \rrbracket_{\mathfrak{A}} = a^{\mathfrak{A}}$  w.r.t. the set-theoretic operators and the aforementioned binary relation operators; i.e.,  $\llbracket t \rrbracket_{\mathfrak{A}}$  is defined as follows:

$$\begin{aligned} \llbracket a \rrbracket_{\mathfrak{A}} &\triangleq a^{\mathfrak{A}} \quad (a \in \Sigma) & \llbracket \perp \rrbracket_{\mathfrak{A}} &\triangleq \emptyset & \llbracket \top \rrbracket_{\mathfrak{A}} &\triangleq |\mathfrak{A}|^2 & \llbracket \text{I} \rrbracket_{\mathfrak{A}} &\triangleq \text{I}_{|\mathfrak{A}|} & \llbracket \text{D} \rrbracket_{\mathfrak{A}} &\triangleq \text{D}_{|\mathfrak{A}|} \\ \llbracket t \cup s \rrbracket_{\mathfrak{A}} &\triangleq \llbracket t \rrbracket_{\mathfrak{A}} \cup \llbracket s \rrbracket_{\mathfrak{A}} & \llbracket t \cap s \rrbracket_{\mathfrak{A}} &\triangleq \llbracket t \rrbracket_{\mathfrak{A}} \cap \llbracket s \rrbracket_{\mathfrak{A}} & \llbracket t^- \rrbracket_{\mathfrak{A}} &\triangleq |\mathfrak{A}|^2 \setminus \llbracket t \rrbracket_{\mathfrak{A}} \\ \llbracket t \cdot s \rrbracket_{\mathfrak{A}} &\triangleq \llbracket t \rrbracket_{\mathfrak{A}} \cdot \llbracket s \rrbracket_{\mathfrak{A}} & \llbracket t \dagger s \rrbracket_{\mathfrak{A}} &\triangleq \llbracket t \rrbracket_{\mathfrak{A}} \dagger \llbracket s \rrbracket_{\mathfrak{A}} & \llbracket t^\pi \rrbracket_{\mathfrak{A}} &\triangleq \llbracket t \rrbracket_{\mathfrak{A}}^\pi. \end{aligned}$$

It is well-known that w.r.t. binary relations, CoR has the same expressive power as the three-variable fragment of first-order logic with equality:<sup>7</sup>

► **Proposition 19** ([28, 29, 11]). *W.r.t. binary relations, the expressive power of  $\mathbf{T}^{\text{CoR}}$  is equivalent to that of the three-variable fragment of first-order logic with equality.*

Let  $\text{REL}$  be the class of all structures. Let  $\text{REL}_{\geq m}$  (resp.  $\text{REL}_{\leq m}$ ) be the class of structures  $\mathfrak{A}$  of  $\#|\mathfrak{A}| \geq m$  (resp.  $\#|\mathfrak{A}| \leq m$ ). For  $\mathcal{C} \subseteq \text{REL}$ , the equivalence relation  $\sim_{\mathcal{C}}$  on  $\mathbf{T}^{\text{CoR}}$  is defined by:  $t \sim_{\mathcal{C}} s \iff \llbracket t \rrbracket_{\mathfrak{A}} = \llbracket s \rrbracket_{\mathfrak{A}}$  for every  $\mathfrak{A} \in \mathcal{C}$ . For  $T \subseteq \mathbf{T}^{\text{CoR}}$ , the equational theory of  $T$  over  $\mathcal{C}$  is the set  $\{\langle t, s \rangle \in T^2 \mid t \sim_{\mathcal{C}} s\}$ . We mainly consider  $\sim_{\text{REL}}$ : the equational theory over  $\text{REL}$ . The following are some instances w.r.t.  $\sim_{\text{REL}}$ :

$$\begin{aligned} a \cdot (b \cdot c) &\sim_{\text{REL}} (a \cdot b) \cdot c & a \cap (b \cup c) &\sim_{\text{REL}} (a \cap b) \cup (a \cap c) & (a^\sim)^\sim &\sim_{\text{REL}} a \\ a \cdot a^\sim &\not\sim_{\text{REL}} a^\sim \cdot a & a \cdot (b \dagger c) &\not\sim_{\text{REL}} (a \cdot b) \dagger c & a^{\{1 \rightarrow 1, 2 \rightarrow 1\}} &\sim_{\text{REL}} (a \cap \text{I}) \cdot \top. \end{aligned}$$

<sup>7</sup> Namely, for every formula  $\varphi$  with two distinct free variables  $z_1, z_2$  in the three-variable fragment of first-order logic with equality, there is  $t \in \mathbf{T}^{\text{CoR}}$  such that for all  $\mathfrak{A}$ ,  $\llbracket \lambda z_1 z_2. \varphi \rrbracket_{\mathfrak{A}} = \llbracket t \rrbracket_{\mathfrak{A}}$ . Conversely, for every  $t \in \mathbf{T}^{\text{CoR}}$ , there is  $\varphi$  such that for all  $\mathfrak{A}$ ,  $\llbracket t \rrbracket_{\mathfrak{A}} = \llbracket \lambda z_1 z_2. \varphi \rrbracket_{\mathfrak{A}}$ . Here,  $\llbracket \lambda z_1 z_2. \varphi \rrbracket_{\mathfrak{A}} \triangleq \{\langle x, y \rangle \in |\mathfrak{A}|^2 \mid \varphi \text{ is true on } \mathfrak{A} \text{ if } z_1, z_2 \text{ are mapped to } x, y, \text{ respectively}\}$ .

The following propositions hold because for each  $m \in \mathbb{N}$ , the number of structures  $\mathfrak{A}$  of  $\#\mathfrak{A} \leq m$  is finite up to isomorphism and each structure is finite.

► **Proposition 20.** *For each  $m \in \mathbb{N}$ ,  $\mathbf{T}^{\text{CoR}}/\sim_{\text{REL}_{\leq m}}$  is finite.*

► **Proposition 21.** *Let  $T \subseteq \mathbf{T}^{\text{CoR}}$  be a subterm-closed set and  $m \geq 1$ . Then,  $T/\sim_{\text{REL}}$  is finite  $\iff T/\sim_{\text{REL}_{\geq m}}$  is finite. Additionally, the equational theory of  $T$  over  $\text{REL}$  is decidable  $\iff$  the equational theory of  $T$  over  $\text{REL}_{\geq m}$  is decidable.*

**Proof.** Because  $t \sim_{\text{REL}} s \iff t \sim_{\text{REL}_{\leq m-1}} s \wedge t \sim_{\text{REL}_{\geq m}} s$ . By Prop. 20 with Prop. 3,  $T/\sim_{\text{REL}_{\leq m-1}}$  is finite and the equational theory of  $T$  over  $\text{REL}_{\leq m-1}$  is decidable. ◀

### 3.2 The dot-dagger alternation hierarchy

► **Definition 22** (the dot-dagger alternation hierarchy [21]). *The sets,  $\{\Sigma_n^{\text{CoR}}, \Pi_n^{\text{CoR}}\}_{n \in \mathbb{N}}$ , are the minimal sets satisfying the following:*

- $\Sigma_0^{\text{CoR}} = \Pi_0^{\text{CoR}} = \{t \in \mathbf{T}^{\text{CoR}} \mid t \text{ does not contain } \cdot \text{ nor } \dagger\}$ ;
- For  $n \geq 0$ ,  $\Sigma_n^{\text{CoR}} \cup \Pi_n^{\text{CoR}} \subseteq \Sigma_{n+1}^{\text{CoR}} \cap \Pi_{n+1}^{\text{CoR}}$ ;
- For  $n \geq 1$ , if  $s, u \in \Sigma_n^{\text{CoR}}$ , then  $s \cup u, s \cap u, s \cdot u, s^\pi \in \Sigma_n^{\text{CoR}}$  and  $s \dagger u \in \Pi_{n+1}^{\text{CoR}}$ ;
- For  $n \geq 1$ , if  $s, u \in \Pi_n^{\text{CoR}}$ , then  $s \cup u, s \cap u, s \dagger u, s^\pi \in \Pi_n^{\text{CoR}}$  and  $s \cdot u \in \Sigma_{n+1}^{\text{CoR}}$ .

For example,  $a \cdot b \in \Sigma_1^{\text{CoR}}$  and  $a \cdot (b \dagger c) \in \Sigma_2^{\text{CoR}}$  (the term  $a \cdot b$  means that for some  $z$ ,  $a(x, z)$  and  $b(z, y)$ ). The term  $a \cdot (b \dagger c)$  means that for some  $z$ , for every  $w$ ,  $a(x, z)$  and  $(b(z, w)$  or  $c(w, y))$ . Here,  $x$  and  $y$  indicate the source and the target, respectively, and each  $a'(x', y')$  denotes that there is an  $a'$ -labelled edge from  $x'$  to  $y'$ ). The dot-dagger alternation hierarchy is an analogy of the quantifier alternation hierarchy in first-order logic (by viewing  $\cdot$  as  $\exists$  and  $\dagger$  as  $\forall$ ). This provides a fine-grained analogy of Prop. 19 w.r.t. the number of quantifier alternations, as follows:

► **Proposition 23** ([21, Cor. 3.14]; cf. Prop. 19). *W.r.t. binary relations, the expressive power of  $\Sigma_n^{\text{CoR}}$  (resp.  $\Pi_n^{\text{CoR}}$ ) is equivalent to that of the level  $\Sigma_n$  (resp.  $\Pi_n$ ) in the quantifier alternation hierarchy of the three-variable fragment of first-order logic with equality.*

Because there are recursive translations for Prop. 23 [21], the following (un-)decidability results follow from those in first-order logic.

► **Proposition 24.** *The equational theory of  $\Sigma_n^{\text{CoR}}$  (resp.  $\Pi_n^{\text{CoR}}$ ) is decidable if  $n \leq 1$  and is undecidable if  $n \geq 2$ .*

**Proof Sketch.** When  $\Sigma$  is a countably infinite set, they follow from the BSR class  $[\exists^* \forall^*, \text{all}] = [3, 27]$  and the reduction class  $[\forall \exists \wedge \forall^3, (\omega, 1)]$  [7, Cor. 3.1.19]. We can strengthen this result even if  $\#\Sigma = 1$  by using a variant of the translation in [19, Lem. 11] for encoding countably infinitely many variables by one variable. (See [23] for more details.) ◀

## 4 On the $k$ -variable-occurrence fragment of $\Sigma_n^{\text{CoR}}$

We now consider  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$ : the  $k$ -variable-occurrence fragment of the level  $\Sigma_n^{\text{CoR}}$  in the dot-dagger alternation hierarchy. Clearly,  $\Sigma_n^{\text{CoR}} = \bigcup_{k \in \mathbb{N}} (\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$ . While the equational theory of  $\Sigma_n^{\text{CoR}}$  is undecidable in general (Prop. 24), we show that the equational theory of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  is decidable (Cor. 26). Our goal in this section is to show the following:

► **Theorem 25.** For each  $n, k \in \mathbb{N}$ ,  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  is finite.

Combining with Prop. 3 yields the following decidability and complexity upper bound. The complexity lower bound is because the equational theory can encode the *boolean sentence value problem* [5] (even if  $n = k = 0$ ), as a given boolean sentence  $\varphi$  is true iff  $t \sim_{\text{REL}} \top$ , where  $t$  is the term obtained from  $\varphi$  by replacing  $\wedge, \vee, \top, \text{F}$  with  $\cap, \cup, \top, \perp$ , respectively.

► **Corollary 26.** For  $n, k \in \mathbb{N}$ , the equational theory of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  over REL is decidable. Moreover, it is complete for DLOGTIME-uniform  $\text{NC}^1$  under DLOGTIME reductions if the input is given as a well-bracketed string.

To prove Thm. 25, we consider the finiteness of  $\mathbf{T}_{(\text{vo} \leq 0)}^{\text{CoR}} / \sim_{\text{REL}}$  in Sect. 4.1 and the finiteness of a monoid for  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  in Sect. 4.2, respectively (cf. Lem. 14).

#### 4.1 On the finiteness of $\mathbf{T}_{(\text{vo} \leq 0)}^{\text{CoR}}$

For the finiteness of  $\mathbf{T}_{(\text{vo} \leq 0)}^{\text{CoR}} / \sim_{\text{REL}}$ , by Prop. 21, it suffices to show the following:

► **Lemma 27.**  $\mathbf{T}_{(\text{vo} \leq 0)}^{\text{CoR}} / \sim_{\text{REL}_{\geq 3}} = \{[\perp]_{\sim_{\text{REL}_{\geq 3}}}, [\top]_{\sim_{\text{REL}_{\geq 3}}}, [I]_{\sim_{\text{REL}_{\geq 3}}}, [D]_{\sim_{\text{REL}_{\geq 3}}}\}$ .

**Proof.** W.r.t.  $\sim_{\text{REL}_{\geq 3}}$ , we prove that the four elements are closed under each operator. For the operators  $\cap, -, \cdot, \smile$ , this is shown by the following Cayley tables:

$\cap$	$\top$	$\perp$	$I$	$D$
$\top$	$\top$	$\perp$	$I$	$D$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$I$	$I$	$\perp$	$I$	$\perp$
$D$	$D$	$\perp$	$\perp$	$D$

$-$	$\diagdown$
$\top$	$\perp$
$\perp$	$\top$
$I$	$D$
$D$	$I$

$\cdot$	$\top$	$\perp$	$I$	$D$
$\top$	$\top$	$\perp$	$\top$	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$I$	$\top$	$\perp$	$I$	$D$
$D$	$\top$	$\perp$	$D$	$\top$

$\smile$	$\diagdown$
$\top$	$\top$
$\perp$	$\perp$
$I$	$I$
$D$	$D$

Note that  $D \cdot D \sim_{\text{REL}_{\geq 3}} \top$  holds thanks to “ $\geq 3$ ”. When  $\#(|\mathfrak{A}|) \geq 3$ , we have:  $\langle x, y \rangle \in \llbracket D \cdot D \rrbracket_{\mathfrak{A}}$  iff  $(\exists z \in |\mathfrak{A}|, z \neq x \wedge z \neq y) \text{ iff } |\mathfrak{A}| \setminus \{x, y\} \neq \emptyset \text{ iff True}$  (cf. Remark 28). (Similarly for  $D \cdot \top \sim_{\text{REL}_{\geq 2}} \top$ .) For the other operators ( $\cup, \dagger, \pi$ ), they can be expressed by using  $\cap, -, \cdot, \smile$  as follows:  $t \cup s \sim_{\text{REL}} (t^- \cap s^-)^-$ ,  $t \dagger s \sim_{\text{REL}} (t^- \cdot s^-)^-$ ,  $t^{\{1 \rightarrow 1, 2 \rightarrow 2\}} \sim_{\text{REL}} t$ ,  $t^{\{1 \rightarrow 1, 2 \rightarrow 1\}} \sim_{\text{REL}} (t \cap I) \cdot \top$ ,  $t^{\{1 \rightarrow 2, 2 \rightarrow 2\}} \sim_{\text{REL}} \top \cdot (t \cap I)$ , and  $t^{\{1 \rightarrow 2, 2 \rightarrow 1\}} = t^\smile$ . Hence, this completes the proof. ◀

► **Remark 28.**  $D \cdot D \not\sim_{\text{REL}} \top$ , whereas  $D \cdot D \sim_{\text{REL}_{\geq 3}} \top$ . For example when  $\#|\mathfrak{A}| = 1$ , since  $\llbracket D \rrbracket_{\mathfrak{A}} = \llbracket \perp \rrbracket_{\mathfrak{A}}$ , we have  $\llbracket D \cdot D \rrbracket_{\mathfrak{A}} = \emptyset \neq |\mathfrak{A}| = \llbracket \top \rrbracket_{\mathfrak{A}}$ . ( $D \cdot D$  is not equivalent to neither one of the four constants w.r.t.  $\sim_{\text{REL}}$ ; thus, there are many constants w.r.t.  $\sim_{\text{REL}}$ .)

#### 4.2 Monoid for $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$

Next, we decompose terms, and then we reduce the finiteness of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  to that of a monoid (cf. Sect. 2.2).

► **Lemma 29.** For each  $n, k \in \mathbb{N}$ , if  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite,  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  is finite.

**Proof.** By specializing  $T$  with  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  and  $\mathcal{C}$  with REL, in Prop. 3 and Lem. 6. ◀

► **Lemma 30.** For each  $n, k \in \mathbb{N}$ ,  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  is finite iff  $(\Pi_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  is finite.

**Proof.**  $\Leftarrow$ : For every term  $t$  in  $(\Pi_n^{\text{CoR}})_{(\text{vo} \leq k)}$ , there is some  $s$  in  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  such that  $t \sim_{\text{REL}} s^-$ . Such  $s$  can be obtained from the term  $t^-$  by taking the complement normal form using the following equations:

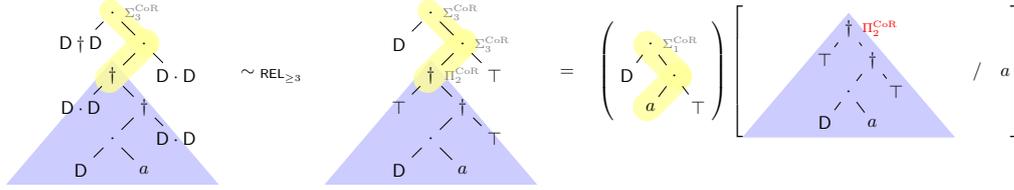
$$\begin{array}{ccccccc} \top^- \sim_{\text{REL}} \perp & & \perp^- \sim_{\text{REL}} \top & & \top^- \sim_{\text{REL}} \text{D} & & \text{D}^- \sim_{\text{REL}} \top \\ (s \cup u)^- \sim_{\text{REL}} s^- \cap u^- & & (s \cap u)^- \sim_{\text{REL}} s^- \cup u^- & & (s^-)^- \sim_{\text{REL}} s & & (s^\pi)^- \sim_{\text{REL}} (s^-)^\pi. \end{array}$$

$\Rightarrow$ : As with  $\Leftarrow$ . ◀

► **Lemma 31** (cf. Lem. 4). Let  $a \in \Sigma$ . For all  $n \geq 2$ ,  $t \in (\Sigma_n^{\text{CoR}})_{(\text{vo} \leq 1)}$ , there are  $t_0 \in (\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)}$  and  $t_1 \in (\Pi_{n-1}^{\text{CoR}})_{(\text{vo} \leq 1)}$  such that  $t \sim_{\text{REL}_{\geq 3}} t_0[t_1/a]$ .

**Proof.** By induction on the pair of  $n$  and  $t$ . We distinguish the following cases. Case  $t \in (\Sigma_{n-1}^{\text{CoR}})_{(\text{vo} \leq 1)}$ : Clear, by IH ( $\cdot: \Pi_{n-2}^{\text{CoR}} \subseteq \Pi_{n-1}^{\text{CoR}}$ ). Case  $t \in (\Pi_{n-1}^{\text{CoR}})_{(\text{vo} \leq 1)}$ : By letting  $t_0 = a$  and  $t_1 = t$ . Case  $t = s \cup u$ : By  $\text{vo}(t) = 1$ ,  $\text{vo}(s) = 0$  or  $\text{vo}(u) = 0$  holds. Sub-case  $\text{vo}(s) = 0$ : By Lem. 27, let  $s' \in \{\perp, \top, \text{I}, \text{D}\}$  be s.t.  $s \sim_{\text{REL}_{\geq 3}} s'$ . By IH w.r.t.  $u$ , let  $u_0 \in (\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)}$ ,  $u_1 \in (\Pi_{n-1}^{\text{CoR}})_{(\text{vo} \leq 1)}$  be s.t.  $u \sim_{\text{REL}_{\geq 3}} u_0[u_1/a]$ . By letting  $t_0 = s' \cup u_0$  and  $t_1 = u_1$ , we have  $t \sim_{\text{REL}_{\geq 3}} t_0[t_1/a]$ . Sub-case  $\text{vo}(u) = 0$ : As with Sub-case  $\text{vo}(s) = 0$ . Case  $t = s \cap u, s \cdot u, s^\pi$ : As with Case  $t = s \cup u$ . ◀

The following is an illustrative example of the decomposition of Lem. 31:



► **Lemma 32.** For each  $n \in \mathbb{N}$ , if  $(\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite,  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite.

**Proof.** By induction on  $n$ . Case  $n \leq 1$ : By the assumption (note that  $(\Sigma_0^{\text{CoR}})_{(\text{vo} \leq 1)} \subseteq (\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)}$ ). Case  $n \geq 2$ : By the assumption,  $(\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite. By IH with Lem. 30,  $(\Pi_{n-1}^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite. Combining them with Lem. 31 (and Prop. 21 for changing  $\sim_{\text{REL}}$  and  $\sim_{\text{REL}_{\geq 3}}$  mutually) yields that  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite. ◀

For  $S \subseteq S^{\text{CoR}}$ , let  $\mathbf{T}^S \subseteq \mathbf{T}^{\text{CoR}}$  be the set of all terms over the signature  $S$ . Then we have:

► **Lemma 33.** If  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}} / \sim_{\text{REL}}$  is finite, then  $(\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite.

**Proof sketch.** Note that  $t, s \in \Sigma_1^{\text{CoR}} ::= u \mid t \cup s \mid t \cap s \mid t \cdot s \mid t^\pi$  (where  $u \in \Sigma_0^{\text{CoR}}$ ,  $\pi \in [1, 2]^{[1, 2]}$ ) and  $u, u' \in \Sigma_0^{\text{CoR}} ::= a \mid u \cup u' \mid u \cap u' \mid u^- \mid \top \mid \perp \mid u^\pi$  (where  $a \in \Sigma$ ,  $\pi \in [1, 2]^{[1, 2]}$ ). By taking the complement ( $-$ ) and projection ( $\pi$ ) normal form and replacing  $\perp$  with  $\text{I} \cap \text{D}$  and  $\top$  with  $\text{I} \cup \text{D}$ , for each  $t \in (\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)}$  and  $a \in \Sigma$ , there are  $t_0 \in \mathbf{T}_{(\text{vo} \leq 1)}^{\{\cup, \cap, \cdot, \text{I}, \text{D}\}}$  and  $t_1 \in \mathbf{T}_{(\text{vo} \leq 1)}^{\{-\} \cup \{\pi \mid \pi \in [1, 2]^{[1, 2]}\}}$  such that  $t \sim_{\text{REL}} t_0[t_1/a]$ . Moreover, by the distributive law of  $\cup$  w.r.t.  $\cdot$  and  $\cap$ , for each  $t \in \mathbf{T}_{(\text{vo} \leq 1)}^{\{\cup, \cap, \cdot, \text{I}, \text{D}\}}$ , there are  $n \in \mathbb{N}$  and  $t_1, \dots, t_n \in \mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}}$  such that  $t \sim_{\text{REL}} t_1 \cup \dots \cup t_n$ . Because  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}} / \sim_{\text{REL}}$  is finite (by the assumption) and  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{-\} \cup \{\pi \mid \pi \in [1, 2]^{[1, 2]}\}} / \sim_{\text{REL}}$  is clearly finite,  $(\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)} / \sim_{\text{REL}}$  is finite. Hence, this completes the proof. (See [23] for more details of the proof.) ◀

## 69:12 On the Finite Variable-Occurrence Fragment of the Calculus of Relations

Combining Lems. 29, 32, and 33 yields that to prove that  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)} / \sim_{\text{REL}}$  is finite, it suffices to prove that  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}} / \sim_{\text{REL}}$  is finite.

Let  $\dot{\Sigma}$  be the set of characters of Def. 7 from the signature  $\{\cap^{(2)}, \cdot^{(2)}, \text{I}^{(0)}, \text{D}^{(0)}, \smile^{(1)}\}$ . That is,  $\dot{\Sigma} \triangleq \{(\_ \cap t), (t \cap \_), (\_ \cdot t), (t \cdot \_) \mid t \in (\mathbf{T}_{(\text{vo} \leq 0)}^{\{\cap, \cdot, \text{I}, \text{D}, \smile\}}) \cup \{\smile\}$ . (While  $\smile$  does not occur in  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}}$ , we introduce  $\smile$  for replacing the primitive character  $(\text{D} \cdot \_)$  with  $\smile (\_ \cdot \text{D})$ . This is not essential but is useful for reducing the number of equations and for simplifying the notation (Def. 35).) Let  $\dot{\sim}_{\text{REL}_{\geq 5}}$  be the equivalence relation on  $\dot{\Sigma}^*$  defined by:  $w \dot{\sim}_{\text{REL}_{\geq 5}} v \iff w[a] \sim_{\text{REL}_{\geq 5}} v[a]$  where  $a \in \Sigma$  is any variable (recall Def. 10).<sup>8</sup>

► **Lemma 34.** *If  $\dot{\Sigma}^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  is finite, then  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}} / \sim_{\text{REL}}$  is finite.*

**Proof.** Since  $\dot{\Sigma}^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  is finite, we have that  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}, \smile\}} / \sim_{\text{REL}_{\geq 5}}$  is finite (Lem. 11); thus,  $\mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, \text{I}, \text{D}\}} / \sim_{\text{REL}_{\geq 5}}$  is finite. Hence by Prop. 21, this completes the proof. ◀

We consider the following finite subset  $\dot{\Sigma}_0$  of  $\dot{\Sigma}$  (cf. Lem. 12):

► **Definition 35.** *Let  $\dot{\Sigma}_0 \subseteq \dot{\Sigma}$  be the finite set  $\{\cap_{\text{I}}, \cap_{\text{D}}, \cdot_{\text{D}}, \smile\}$ , where  $\cap_{\text{I}}, \cap_{\text{D}}, \cdot_{\text{D}}$  are abbreviations of  $(\_ \cap \text{I}), (\_ \cap \text{D}), (\_ \cdot \text{D})$ , respectively.*

► **Lemma 36.** *If  $\dot{\Sigma}_0^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  is finite, then  $\dot{\Sigma}^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  is finite.*

**Proof.** It suffices to prove the following: for every  $a \in \dot{\Sigma}$ , there is  $w \in \dot{\Sigma}_0^*$  such that  $a \dot{\sim}_{\text{REL}_{\geq 5}} w$ . Case  $a = (\_ \cap t), (\_ \cdot t)$ : Since  $\text{vo}(t) = 0$ , by using Lem. 27, they are shown by distinguishing the following four sub-cases, as follows:

	$t \sim_{\text{REL}_{\geq 3}} \perp$	$t \sim_{\text{REL}_{\geq 3}} \top$	$t \sim_{\text{REL}_{\geq 3}} \text{I}$	$t \sim_{\text{REL}_{\geq 3}} \text{D}$
$a = (\_ \cap t)$	$\cap_{\text{I}} \cap_{\text{D}}$	$\varepsilon$	$\cap_{\text{I}}$	$\cap_{\text{D}}$
$a = (\_ \cdot t)$	$\cap_{\text{I}} \cap_{\text{D}}$	$\cdot_{\text{D}} \cdot_{\text{D}}$	$\varepsilon$	$\cdot_{\text{D}}$

Case  $a = (t \cap \_), (t \cdot \_)$ : By  $(t \cap \_) = \smile (\_ \cap t)$  and applying the above case analysis for  $(\_ \cap t)$ , this case can be proved (similarly for  $(t \cdot \_)$ ). Case  $a = \smile$ : Since  $\smile \in \dot{\Sigma}_0$ . ◀

Thus, our goal is to prove that  $\dot{\Sigma}_0^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  is finite.

### 4.3 On the finiteness of the monoid

For the finiteness of  $\dot{\Sigma}_0^* / \dot{\sim}_{\text{REL}_{\geq 5}}$  (cf. Lem. 15), we present the 21 equations in Fig. 1.<sup>9</sup> For  $i \in [1, 21]$ , let  $w_i, v_i$  be words such that  $w_i = v_i$  denotes the  $i$ -th equation.

► **Lemma 37** (soundness). *For each  $i \in [1, 21]$ ,  $w_i \dot{\sim}_{\text{REL}_{\geq 5}} v_i$ .*

**Proof Sketch.** We prove  $w_i[a] \sim_{\text{REL}_{\geq 5}} v_i[a]$ , where  $a$  is any variable. This equation can be translated to the validity of a first-order sentence via the standard translation [28]. Here, we add the formula  $\exists x_1, \dots, x_5, \wedge_{i,j \in [1,5]; i \neq j} x_i \neq x_j$  as an axiom, for forcing  $\dot{\sim}_{\text{REL}_{\geq 5}}$ . Thanks to this encoding, each of them can also be tested by using ATP/SMT systems. Nevertheless, in

<sup>8</sup> The condition “ $\geq 5$ ” is needed for some equations in Fig. 1.

<sup>9</sup> The most technical part of the paper is to collect these equations. They are obtained by running a program based on Algorithm 1 using ATP/SMT systems.



**Proof of Thm. 25.** We have:  $\dot{\Sigma}_0^*/\sim_{\text{REL}_{\geq 5}}$  is finite (Lem. 39)  $\implies \dot{\Sigma}^*/\sim_{\text{REL}_{\geq 5}}$  is finite (Lem. 36)  $\implies \mathbf{T}_{(\text{vo} \leq 1)}^{\{\cap, \cdot, !, \text{D}\}}/\sim_{\text{REL}}$  is finite (Lem. 34)  $\implies (\Sigma_1^{\text{CoR}})_{(\text{vo} \leq 1)}/\sim_{\text{REL}}$  is finite (Lem. 33)  $\implies (\Sigma_n^{\text{CoR}})_{(\text{vo} \leq 1)}/\sim_{\text{REL}}$  is finite (Lem. 32)  $\implies (\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}/\sim_{\text{REL}}$  is finite (Lem. 29).  $\blacktriangleleft$

$\blacktriangleright$  Remark 40. The finite axiomatizability of the equational theory of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  over REL immediately follows from the finiteness of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}/\sim_{\text{REL}}$ .

## 5 Conclusion

We have introduced the *k-variable-occurrence fragment* and presented an approach for showing the decidability of the equational theory from the finiteness. As a case study, we have proved that the equational theory of  $(\Sigma_n^{\text{CoR}})_{(\text{vo} \leq k)}$  is decidable, whereas that of  $\Sigma_n^{\text{CoR}}$  is undecidable in general. We leave the decidability open for the equational theory of CoR with *full dot-dagger alternation* (i.e.,  $\mathbf{T}_{(\text{vo} \leq k)}^{\text{CoR}}$ , in this paper). Our approach may apply to some other algebras/logics. It would be interesting to consider the finite variable-occurrence fragment for other systems (e.g., CoR with antidomain [13, 10], dynamic logics [12], relation algebras). It would also be interesting to extend our result to *first-order logic with equality* (cf. Prop. 19) – for example, is the *k-atomic-predicate-occurrence fragment* of the *m-variable fragment* of first-order logic with equality decidable?

---

## References

- 1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
- 2 Hajnal Andréka and D. A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.
- 3 Paul Bernays and Moses Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99(1):342–372, 1928. doi:10.1007/BF01459101.
- 4 Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Springer, 1993. doi:10.1007/978-1-4613-9771-7.
- 5 Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In *STOC*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409.
- 6 Samuel R. Buss, Stepane A. Cook, Anshul Gupta, and Vijaya Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992. doi:10.1137/0221046.
- 7 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer, 1997. URL: <https://link.springer.com/9783540423249>.
- 8 Alonzo Church. A note on the Entscheidungsproblem. *The Journal of Symbolic Logic*, 1(01):40–41, 1936. doi:10.2307/2269326.
- 9 Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 13243 of *LNCS*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3\_24.
- 10 Jules Desharnais, Peter Jipsen, and Georg Struth. Domain and antidomain semigroups. In *RelMiCS*, volume 5827 of *LNCS*, pages 73–87. Springer, 2009. doi:10.1007/978-3-642-04639-1\_6.
- 11 Steven Givant. The calculus of relations as a foundation for mathematics. *Journal of Automated Reasoning*, 37(4):277–322, 2007. doi:10.1007/s10817-006-9062-x.
- 12 David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. The MIT Press, 2000. doi:10.7551/mitpress/2516.001.0001.

- 13 Marco Hollenberg. An equational axiomatization of dynamic negation and relational composition. *Journal of Logic, Language and Information*, 6(4):381–401, 1997. doi:10.1023/A:1008271805106.
- 14 A. S. Kahr, Edward F. Moore, and Hao Wang. Entscheidungsproblem reduced to the AEA case. *Proceedings of the National Academy of Sciences*, 48(3):365–377, 1962. doi:10.1073/pnas.48.3.365.
- 15 Markus Lohrey. On the parallel complexity of tree automata. In *RTA*, volume 2051 of *LNCS*, pages 201–215. Springer, 2001. doi:10.1007/3-540-45127-7\_16.
- 16 Roger D. Maddux. Undecidable semiassociative relation algebras. *The Journal of Symbolic Logic*, 59(02):398–418, 1994. doi:10.2307/2275397.
- 17 A. A. Markov. On the impossibility of certain algorithms in the theory of associative systems. *Doklady Akademii Nauk SSSR* 55, 58, pages 587–590, 353–356, 1947.
- 18 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within  $NC^1$ . *Journal of Computer and System Sciences*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
- 19 Yoshiki Nakamura. The undecidability of FO3 and the calculus of relations with just one binary relation. In *ICLA*, volume 11600 of *LNCS*, pages 108–120. Springer, 2019. doi:10.1007/978-3-662-58771-3\_11.
- 20 Yoshiki Nakamura. Expressive power and succinctness of the positive calculus of relations. In *RAMiCS*, volume 12062 of *LNCS*, pages 204–220. Springer, 2020. doi:10.1007/978-3-030-43520-2\_13.
- 21 Yoshiki Nakamura. Expressive power and succinctness of the positive calculus of binary relations. *Journal of Logical and Algebraic Methods in Programming*, 127:100760, 2022. doi:10.1016/j.jlamp.2022.100760.
- 22 Yoshiki Nakamura. Existential calculi of relations with transitive closure: Complexity and edge saturations. In *LICS*, pages 1–13. IEEE, 2023. doi:10.1109/LICS56636.2023.10175811.
- 23 Yoshiki Nakamura. On the finite variable-occurrence fragment of the calculus of relations with bounded dot-dagger alternation, 2023. doi:10.48550/arXiv.2307.05046.
- 24 Yoshiki Nakamura. Repository for “on the finite variable-occurrence fragment of the calculus of relations with bounded dot-dagger alternation”, 2023. URL: [https://bitbucket.org/yoshikinakamura/k-vo\\_cor\\_with\\_bounded\\_dot\\_dagger/](https://bitbucket.org/yoshikinakamura/k-vo_cor_with_bounded_dot_dagger/).
- 25 Emil L. Post. Recursive unsolvability of a problem of Thue. *The Journal of Symbolic Logic*, 12(1):1–11, 1947. doi:10.2307/2267170.
- 26 Damien Pous. On the positive calculus of relations with transitive closure. In *STACS*, volume 96 of *LIPICs*, pages 3:1–3:16. Schloss Dagstuhl, 2018. doi:10.4230/LIPICs.STACS.2018.3.
- 27 F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930. doi:10.1112/plms/s2-30.1.264.
- 28 Alfred Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941. doi:10.2307/2268577.
- 29 Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables*, volume 41. American Mathematical Society, 1987. doi:10.1090/coll/041.
- 30 Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. doi:10.1112/plms/s2-42.1.230.



# Effective Continued Fraction Dimension Versus Effective Hausdorff Dimension of Reals

Satyadev Nandakumar ✉ 🏠 

Department of Computer Science, Indian Institute of Technology Kanpur, India

Akhil S ✉ 🏠 

Department of Computer Science, Indian Institute of Technology Kanpur, India

Prateek Vishnoi ✉ 🏠 

Department of Computer Science, Indian Institute of Technology Kanpur, India

---

## Abstract

We establish that constructive continued fraction dimension originally defined using  $s$ -gales [20] is robust, but surprisingly, that the effective continued fraction dimension and effective (base- $b$ ) Hausdorff dimension of the same real can be unequal in general.

We initially provide an equivalent characterization of continued fraction dimension using Kolmogorov complexity. In the process, we construct an optimal lower semi-computable  $s$ -gale for continued fractions. We also prove new bounds on the Lebesgue measure of continued fraction cylinders, which may be of independent interest.

We apply these bounds to reveal an unexpected behavior of continued fraction dimension. It is known that feasible dimension is invariant with respect to base conversion [8]. We also know that Martin-Löf randomness and computable randomness are invariant not only with respect to base conversion, but also with respect to the continued fraction representation [20]. In contrast, for any  $0 < \varepsilon < 0.5$ , we prove the existence of a real whose effective Hausdorff dimension is less than  $\varepsilon$ , but whose effective continued fraction dimension is greater than or equal to 0.5. This phenomenon is related to the “non-faithfulness” of certain families of covers, investigated by Peres and Torbin [22] and by Alberverio, Ivanenko, Lebid and Torbin [1].

We also establish that for any real, the constructive Hausdorff dimension is at most its effective continued fraction dimension.

**2012 ACM Subject Classification** Theory of computation → Computability; Mathematics of computing → Information theory

**Keywords and phrases** Algorithmic information theory, Kolmogorov complexity, Continued fractions, Effective Hausdorff dimension

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.70

**Acknowledgements** The authors would like to thank Subin Pulari for his comments and helpful discussions.

## 1 Introduction

The concept of an individual random sequence, first defined by Martin-Löf using constructive measure [15], is well-established and mathematically robust - very different approaches towards the definition identify precisely the same sequences as random. These include Kolmogorov incompressibility (Levin [10], Chaitin [3]) and unpredictability by martingales [24]. While the theory of Martin-Löf randomness *classifies* sequences into random and non-random, it does not *quantify* the information rate in a non-random sequence. Lutz effectivized the classical notions of Hausdorff and packing dimensions [12], surprisingly extending it to individual infinite binary sequences [13], yielding a notion of information density in sequences. This definition also has several equivalent definitions in terms of Kolmogorov compression



© Satyadev Nandakumar, Akhil S, and Prateek Vishnoi;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 70; pp. 70:1–70:15

Leibniz International Proceedings in Informatics



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

rates [16], unpredictability by  $s$ -gales, and using covers [12, 13]. These definitions have led to a rich variety of applications in various domains of computability and complexity theory (see for example, Downey and Hirschfeldt [5], Nies [21]).

Recently, settings more general than the Cantor space of infinite binary (or in general, infinite sequences from a finite alphabet) have been studied by Lutz and Mayordomo [14], and Mayordomo [17, 18]. Prominent among them is Mayordomo’s definition of effective Hausdorff dimension for a very general class of metric spaces [17, 18]. Nandakumar and Vishnoi [20] and Vishnoi [28] define the notion of effective dimension of continued fractions, which involves a countably infinite alphabet, and is thus a setting which cannot be studied using Mayordomo’s framework. This latter setting is interesting topologically since the space of continued fractions is non-compact, and interesting measure-theoretically since the natural shift invariant measure, the Gauss measure, is a non-product measure.

Nandakumar and Vishnoi [20] use the notion of an  $s$ -gale on the space of continued fractions to define effective dimension. Vishnoi [28] introduced the notion of Kolmogorov complexity of finite continued fraction strings using a one to one binary encoding. Vishnoi [28] also shows that the notion of Kolmogorov complexity is invariant under computable 1-1 encodings, upto an additive constant.

In this work, we first establish the mathematical robustness of the notion of effective dimension, by proving an equivalent characterization using Kolmogorov complexity of continued fractions. The characterization achieves the necessary equivalence by choosing a binary encoding of continued fractions which has a compelling geometric intuition, and then applying Mayordomo’s characterization of effective (binary) Hausdorff dimension using Kolmogorov complexity [16]. In the process, analogous to the notion of an optimal constructive supergale on the Cantor space defined by Lutz [13], we provide the construction of a lower semi-computable  $s$ -gale that is optimal for continued fractions. We also prove new bounds on the Lebesgue measure of continued fraction cylinders using the digits of the continued fraction expansion, a result which may be of independent interest.

The topological and measure-theoretic intricacies involved in this setting imply that some, but not all, “natural” properties of randomness and dimension carry over from the binary setting. For example, while Martin-Löf and computable randomness are invariant with respect to the conversion between the base- $b$  and continued fraction expansion of the same real [19, 20], Vandehey [26] and Scheerer [23] show that other notions of randomness like absolute normality and normality for continued fractions are not identical.

Staiger [25] showed that the Kolmogorov complexity of a base  $b$  expansion of a real  $\alpha, 0 \leq \alpha \leq 1$ , is independent of the chosen base  $b$ . Aligning with this, Hitchcock and Mayordomo [8] establish that feasible dimension of a real is the same when converting between one base to another. Hitherto, it was unknown whether effective dimension is invariant with respect to conversion between base- $b$  and continued fraction representations. Since we can convert between the representations efficiently, it is possible that these are equal. We show this is true in one direction, that the effective base  $b$  dimension is a lower bound for effective continued fraction dimension.

However, using the technique of diagonalization against the optimal lower semicomputable continued fraction  $s$ -gale and using set covering techniques used in recent works by Peres and Torbin [22], Albeverio, Ivanenko, Lebid and Torbin [1] and Albeverio, Kondratiev, Nikiforov and Torbin [2] to show the “non-faithfulness” of certain families of covers, we show that the reverse direction does not hold, in general. We prove the following result: for every  $0 < \varepsilon < 0.5$ , there is a real whose effective (binary) Hausdorff dimension is less than  $\varepsilon$  while its effective continued fraction dimension is at least 0.5. By the result of Hitchcock and

Mayordomo [8], this also implies that the effective base- $b$  dimension of this real is less than  $\varepsilon$  in every base- $b$ ,  $b \geq 2$ . Thus, surprisingly, there is a sharp gap between the effective (base- $b$ ) dimension of a real and its effective continued fraction dimension, highlighting another significant difference in this setting.

## 2 Preliminaries

We denote the binary alphabet by  $\Sigma$ . The set of strings of a particular length  $n$  is denoted  $\Sigma^n$ . The set of all finite binary strings is denoted  $\Sigma^*$  and infinite binary sequences is denoted  $\Sigma^\infty$ . For a binary string  $v \in \Sigma^n \setminus \{0^n \cup 1^n\}$ ,  $v - 1$  denotes the string occurring just before  $v$  lexicographically, and  $v + 1$  the string occurring just after  $v$  lexicographically. We use  $\mathbb{N}$  to denote the set of positive integers. The set of finite continued fractions is denoted  $\mathbb{N}^*$  and the set of all infinite continued fractions, as  $\mathbb{N}^\infty$ .

We adopt the notation  $[a_1, a_2, \dots]$  for the continued fraction

$$\frac{1}{a_1 + \frac{1}{a_2 + \dots}}$$

and similarly,  $[a_1, a_2, \dots, a_n]$  for finite continued fractions.

If a finite binary string  $x$  is a prefix of a finite string  $z$  or an infinite binary sequence  $Z$ , then we denote this by  $x \sqsubseteq z$  or  $x \sqsubseteq Z$  respectively. If  $x$  is a proper prefix of a finite string  $z$ , we denote it by  $x \sqsubset z$ . We adopt the same notation for denoting that a finite continued fraction  $v$  is a prefix of another continued fraction. For a  $v \in \mathbb{N}^*$ , the *cylinder set* of  $v$ , denoted  $C_v$ , is defined by  $C_v = \{Y \in \mathbb{N}^\infty \mid v \sqsubset Y\}$ . For a  $w \in \Sigma^*$ ,  $C_w$  is defined similarly. For a continued fraction string  $v = [a_1 \dots a_n]$ ,  $P(v)$  denotes the string  $[a_1, \dots, a_{n-1}]$ .  $\lambda$  denotes the empty string and we define  $P(\lambda) = \lambda$ .

For  $v \in \mathbb{N}^*$ ,  $\mu(v)$  refers to the Lebesgue measure of the continued fraction cylinder  $C_v$ .  $\gamma(v)$  refers to the Gauss measure of the continued fraction cylinder  $C_v$ , defined by  $\gamma(v) = \int_{C_v} \frac{1}{1+x} dx$ . We use the same notation for a binary cylinder  $w \in \Sigma^*$ . It is well-known that the Gauss measure is absolutely continuous with respect to the Lebesgue measure, and is invariant with respect to the left-shift transformation on continued fractions (see for example, [4], or [6]). Wherever there is no scope for confusion, for a  $v \in \mathbb{N}^*$ , we use  $\mu(v)$  and  $\gamma(v)$  to represent  $\mu(C_v)$  and  $\gamma(C_v)$  respectively. The same holds for a  $v \in \Sigma^*$ . We also use the notation  $\mu^s(v)$  and  $\gamma^s(v)$  to denote  $(\mu(v))^s$  and  $(\gamma(v))^s$  respectively. For a continued fraction string  $v = [a_1, \dots, a_n]$ , we call  $n$  as the rank of  $v$ , and we denote it using  $\text{rank}(v)$ .  $[v, i]$  denotes the continued fraction  $[a_1, \dots, a_n, i]$ . For an infinite continued fraction string  $Y = [a_1, a_2, \dots]$ ,  $Y \upharpoonright n$  denotes the continued fraction string corresponding to the first  $n$  entries of  $Y$ , that is  $Y \upharpoonright n = [a_1, a_2 \dots a_n]$ . For  $k \in \mathbb{N}$ ,  $\mathbb{N}^{\leq k}$  refers to the set of continued fraction strings having rank less than or equal to  $k$ . All logarithms in the work have base 2, unless specified otherwise. For any sets  $A$  and  $B$ ,  $A \Delta B$  denotes the symmetric set difference operator, defined by  $(A \setminus B) \cup (B \setminus A)$ . In this work, for ease of notation,  $Y \in \mathbb{N}^*$  denotes an infinite continued fraction and  $X \in \Sigma^\infty$  denotes an infinite binary sequence.

### 2.1 Constructive dimension of binary sequences

Lutz [13] defines the notion of effective (equivalently, constructive) dimension of an individual infinite binary sequence using the notion of the success of  $s$ -gales.

► **Definition 1** (Lutz [13]). For  $s \in [0, \infty)$ , a binary  $s$ -gale is a function  $d : \Sigma^* \rightarrow [0, \infty)$  such that  $d(\lambda) < \infty$  and for all  $w \in \Sigma^*$ ,  $d(w)[\mu(C_w)]^s = \sum_{i \in \{0,1\}} d(wi)[\mu(C_{wi})]^s$ .

The success set of  $d$  is  $S^\infty(d) = \left\{ X \in \mathbb{N}^\infty \mid \limsup_{n \rightarrow \infty} d(X \upharpoonright n) = \infty \right\}$ .

For  $\mathcal{F} \subseteq [0, 1]$ ,  $\mathcal{G}(\mathcal{F})$  denotes the set of all  $s \in [0, \infty)$  such that there exists a lower semicomputable binary  $s$ -gale  $d$  with  $\mathcal{F} \subseteq S^\infty(d)$ .

The constructive dimension or effective Hausdorff dimension of  $\mathcal{F} \subseteq [0, 1]$  is  $\text{cdim}(\mathcal{F}) = \inf \mathcal{G}(\mathcal{F})$  and the constructive dimension of a sequence  $X \in \Sigma^\infty$  is  $\text{cdim}(X) = \text{cdim}(\{X\})$ .

### 3 Effective Continued Fraction Dimension using $s$ -gales

Nandakumar and Vishnoi [20] formulate the notion of effective dimension of continued fractions using the notion of lower semicomputable continued fraction  $s$ -gales. Whereas a binary  $s$ -gale bets on the digits of the binary expansion of a number, a continued fraction  $s$ -gale places bets on the digits of its continued fraction expansion.

► **Definition 2** (Nandakumar, Vishnoi [20]). For  $s \in [0, \infty)$ , a continued fraction  $s$ -gale is a function  $d : \mathbb{N}^* \rightarrow [0, \infty)$  such that  $d(\lambda) < \infty$  and for all  $w \in \mathbb{N}^*$ , the following holds.

$$d(w)[\gamma(C_w)]^s = \sum_{i \in \mathbb{N}} d(wi)[\gamma(C_{wi})]^s.$$

The success set of  $d$  is  $S^\infty(d) = \left\{ Y \in \mathbb{N}^\infty \mid \limsup_{n \rightarrow \infty} d(Y \upharpoonright n) = \infty \right\}$ .

In this paper, we deal with the notion of effective or equivalently, constructive dimension. In order to effectivize the notion of  $s$ -gales, we require them to be *lower semicomputable*.

► **Definition 3.** A function  $d : \mathbb{N}^* \rightarrow [0, \infty)$  is called lower semicomputable if there exists a total computable function  $\hat{d} : \mathbb{N}^* \times \mathbb{N} \rightarrow \mathbb{Q} \cap [0, \infty)$  such that the following two conditions hold.

- **Monotonicity:** For all  $w \in \mathbb{N}^*$  and for all  $n \in \mathbb{N}$ , we have  $\hat{d}(w, n) \leq \hat{d}(w, n+1) \leq d(w)$ .
- **Convergence:** For all  $w \in \mathbb{N}^*$ ,  $\lim_{n \rightarrow \infty} \hat{d}(w, n) = d(w)$ .

For  $\mathcal{F} \subseteq [0, 1]$ ,  $\mathcal{G}_{CF}(\mathcal{F})$  denotes the set of all  $s \in [0, \infty)$  such that there exists a lower semicomputable continued fraction  $s$ -gale  $d$  with  $\mathcal{F} \subseteq S^\infty(d)$ .

► **Definition 4** (Nandakumar, Vishnoi [20]). The effective continued fraction dimension of  $\mathcal{F} \subseteq [0, 1]$  is

$$\text{cdim}_{CF}(\mathcal{F}) = \inf \mathcal{G}_{CF}(\mathcal{F}).$$

The effective continued fraction dimension of a sequence  $Y \in \mathbb{N}^\infty$  is defined by  $\text{cdim}_{CF}(\{Y\})$ , the effective continued fraction dimension of the singleton set containing  $Y$ .

#### 3.1 Conversion of binary $s$ -gales into continued fraction $s$ -gales

In this subsection, from a continued fraction  $s'$ -gale  $d : \mathbb{N}^* \rightarrow [0, \infty)$ , for any  $s > s'$ , we construct a binary  $s$ -gale  $h : \Sigma^* \rightarrow [0, \infty)$  which succeeds on all the reals on which  $d$  succeeds. The construction proceeds in multiple steps. We first mention some technical lemmas which we use in the proof.

The following lemma is an easy consequence of the fact that the Gauss measure is absolutely continuous with respect to the Lebesgue measure (see for example, Nandakumar and Vishnoi [20]).

► **Lemma 5.** For any interval  $B \subseteq (0, 1)$ , we have

$$\frac{1}{2 \ln 2} \mu(B) \leq \gamma(B) \leq \frac{1}{\ln 2} \mu(B).$$

In the construction that follows, we formulate betting strategies on binary cylinders based on continued fraction cylinders. In order to do this conversion, we require the following bounds on the relationships between the lengths of continued fraction cylinders and binary cylinders.

► **Lemma 6** (Nandakumar, Vishnoi [20]). For any  $0 \leq a < b \leq 1$ , let  $[\frac{m}{2^k}, \frac{m+1}{2^k})$ , where  $0 \leq m \leq 2^k - 1$  be one of the largest dyadic intervals which is a subset of  $[a, b)$ , then  $\frac{1}{2^k} \geq \frac{1}{4}(b - a)$ .

► **Lemma 7** (Falconer [7]). For any  $0 \leq a < b \leq 1$ , let  $[\frac{m}{2^k}, \frac{m+1}{2^k})$ ,  $[\frac{m+1}{2^k}, \frac{m+2}{2^k})$ , where  $0 \leq m \leq 2^k - 2$ , be the smallest consecutive dyadic intervals whose union covers  $[a, b)$ . Then  $\frac{1}{2^k} \leq 2(b - a)$ .

The following lemma is a generalization of the Kolmogorov inequality for continued fraction martingales (Vishnoi [27]) to  $s$ -gales. The lemma states that an equality holds in the case of decompositions using prefix-free subcylinder sets upto a finite depth.

► **Lemma 8.** Let  $d : \mathbb{N}^* \rightarrow [0, \infty)$  be a continued fraction  $s$ -gale. Let  $v \in \mathbb{N}^*$  and for some  $k \in \mathbb{N}$ , let  $A$  be a prefix free set of elements in  $\mathbb{N}^{\leq k}$  such that  $\cup_{w \in A} C_w = C_v$ . Then, we have  $d(v)\gamma^s(v) = \sum_{w \in A} d(w)\gamma^s(w)$ .

In the construction of a binary  $s$ -gale from continued fraction gales, The first step is the following decomposition of a binary cylinder into a set of prefix free continued fraction cylinders.

► **Lemma 9** (Vishnoi [27]). For every  $w \in \Sigma^*$ , there exists a set  $I(w) \subseteq \mathbb{N}^*$  and a constant  $k \in \mathbb{N}$  such that,

1.  $y \in \mathbb{N}^{\leq k}$  for every  $y \in I(w)$ .
2.  $(\cup_{y \in I(w)} C_y) \Delta C_w \subseteq \{\inf(C_w), \sup(C_w)\}$
3.  $I(w0) \cup I(w1) = I(w)$
4.  $I(w0) \cap I(w1) = \phi$

Moreover, given  $w \in \Sigma^*$ , Vishnoi [27] gives a division algorithm to compute  $I(w)$ . It is also clear from the division algorithm that for all  $w \in \Sigma^*$ , there exists a  $u \in I(w)$  such that for all  $v \in (I(w0) \cup I(w1)) \setminus I(w)$ , we have  $u \sqsubset v$ . This  $u \in I(w)$  is the continued fraction cylinder for which the mid point of  $w$ ,  $m(w)$  is an interior point in  $C_u$  and therefore gets divided.

From a continued fraction martingale, Vishnoi [27] uses the decomposition  $I(w)$  to construct a binary martingale that places the same aggregate bets on an interval. We generalize this construction to the setting of  $s$ -gales. Given a continued fraction  $s'$ -gale  $d : \mathbb{N}^* \rightarrow [0, \infty)$ , using the decomposition  $I(w)$ , we construct a binary  $s'$ -gale  $H_d$  from  $d$ .

► **Definition 10.** Given any continued fraction  $s'$ -gale  $d : \mathbb{N}^* \rightarrow [0, \infty)$ , define the Proportional binary  $s'$ -gale of  $d$ ,  $H_d : \Sigma^* \rightarrow [0, \infty)$  as follows:

$$H_d(w) = \sum_{y \in I(w)} d(y) \left( \frac{\gamma(y)}{\mu(w)} \right)^{s'}.$$

## 70:6 Effective Continued Fraction Dimension

For a  $w \in \Sigma^*$ , let  $I'(w) = I(w0) \cup I(w1)$ . Then we have,

$$H_d(w0) + H_d(w1) = 2^{s'} \sum_{y \in I'(w)} d(y) \left( \frac{\gamma(y)}{\mu(w)} \right)^{s'}.$$

Let  $u \in I(w)$  such that for all  $v \in I'(w) \setminus I(w)$ ,  $u \sqsubset v$ . Hence, by Lemma 8, it follows that  $\sum_{y \in I'(w)} d(y) \gamma^{s'}(y) = \sum_{y \in I(w)} d(y) \gamma^{s'}(y)$ . Therefore, we have  $H_d(w0) + H_d(w1) = 2^{s'} H_d(w)$ , so  $H_d$  is an  $s'$ -gale. Also as  $\gamma(\lambda) = 1$ , we have that  $H_d(\lambda) = d(\lambda)$ .

As  $I(w)$  is computably enumerable, it follows that  $H_d$  is lower semicomputable if  $d$  is lower semicomputable.

The construction by Vishnoi [27] proceeds using the *savings-account trick* for martingales. In the setting of  $s$ -gales, however, the concept of a savings account does not work directly. Therefore, we require additional constructions in this setting.

Using ideas from the construction given in Lemma 3.1 in Hitchcock and Mayordomo [8], we construct a “smoothed”  $s$ -gale  $H_h : \Sigma^* \rightarrow [0, \infty)$  from the proportional  $s'$ -gale constructed in Definition 10.

► **Definition 11.** For a  $w \in \Sigma^*$ , and an  $n > |w|$ , we define

$$\begin{aligned} F_n(w) &= \{u \in \{0^n \cup 1^n\} \mid w \sqsubseteq u\} \cup \{u \in \Sigma^n \setminus \{0^n \cup 1^n\} \mid w \sqsubseteq u + 1 \text{ and } w \sqsubseteq u - 1\}, \\ H_n(w) &= \{u \in \Sigma^n \mid w \sqsubseteq u \text{ or } w \sqsubseteq u + 1 \text{ or } w \sqsubseteq u - 1\} \setminus F_n. \end{aligned}$$

► **Definition 12.** Given an  $s'$ -gale  $h : \Sigma^* \rightarrow [0, \infty)$ , for any  $s > s'$  and for each  $n \in \mathbb{N}$ , define:

$$h_n(w) = \begin{cases} 2^{s|w|} \left( \sum_{u \in H_n(w)} \frac{1}{2} h(u) + \sum_{u \in F_n(w)} h(u) \right) & \text{if } |w| < n \\ 2^{(s-1)(|w|-n+1)} h_n(w[0 \dots n-2]) & \text{otherwise.} \end{cases}$$

Define  $S_h : \Sigma^* \rightarrow [0, \infty)$  by

$$S_h(w) = \sum_{n=0}^{\infty} 2^{-sn} h_n(w).$$

We call  $S_h$  as the smoothed  $s$ -gale of  $h$ .

Consider a string  $w \in \Sigma^n$  other than  $0^n$  and  $1^n$ . In  $h_n$ , a factor of half the capital of  $w$  gets assigned to its immediate parent  $w'$ . The other half is assigned to the neighbor of  $w'$  to which  $w$  is adjacent to.

It is straightforward to verify that each  $h_n$  is an  $s$ -gale.  $S_h$  is a combination of  $s$ -gales, and hence is a valid  $s$ -gale. Note that  $h_n(\lambda) = \sum_{u \in \Sigma^n} h(u) = 2^{s'n}$ . Therefore as  $s > s'$ ,  $S_h(\lambda) = \sum_{n \in \mathbb{N}} 2^{(s'-s)n}$  is finite. If  $h$  is lower semicomputable, it follows that  $S_h$  is lower semicomputable.

Combining the constructions given in the section, for any  $s > s'$ , we show the construction of a binary  $s$ -gale from a continued fraction  $s'$ -gale, satisfying certain bounds on the capital acquired.

This construction helps to establish a lower bound on effective continued fraction dimension using effective binary dimension. It is also central in formulating a Kolmogorov complexity characterization for continued fraction dimension.

► **Lemma 13.** For  $s' \in (0, \infty)$ , let  $d : \mathbb{N}^* \rightarrow [0, \infty)$  be a continued fraction  $s'$ -gale. Then, for any  $s > s'$ , there exists a binary  $s$ -gale  $h : \Sigma^* \rightarrow [0, \infty)$  such that for any  $v \in \mathbb{N}^*$  and for any  $b \in \Sigma^*$  such that  $C_b \cap C_v \neq \emptyset$  and  $\frac{1}{16}\mu(v) \leq \mu(b) \leq 2\mu(v)$ , we have

$$h(b) \geq c_s d(v),$$

where  $c_s$  is a constant that depends on  $s$ . Moreover, if  $d$  is lower semicomputable, then  $h$  is lower semicomputable.

#### 4 Kolmogorov Complexity characterization of Continued Fraction Dimension

Mayordomo [16] extended the result by Lutz [11] to show that effective dimension of a binary sequence  $X \in \Sigma^\infty$  can be characterized in terms of the Kolmogorov complexity of the finite prefixes of  $X$ .

► **Theorem 14** (Mayordomo [16] and Lutz [11]). For every  $X \in \Sigma^\infty$ ,

$$\text{cdim}(X) = \liminf_{n \rightarrow \infty} \frac{K(X \upharpoonright n)}{n}.$$

We provide a similar characterization for effective continued fraction dimension. To obtain the Kolmogorov complexity of a continued fraction string, we use the Kolmogorov complexity of one of its binary encodings.

The idea of encoding a finite continued fraction using a 1-1 binary encoding is present in Vishnoi [28]. The author presents an invariance theorem stating that every computable binary 1-1 encoding of continued fractions defines the same Kolmogorov complexity, up to an additive constant. Hence in this work, we use a new binary encoding to define Kolmogorov complexity of continued fractions, which helps us establish the characterization of effective dimension of continued fractions in a fairly simple manner while having intuitive geometric meaning.

► **Definition 15** (Many-one binary encoding). For a continued fraction string  $v \in \mathbb{N}^*$ , let  $b_v$  be the leftmost maximal binary cylinder which is enclosed by  $C_v$ . We define  $E(v) = b_v$ .

► **Lemma 16.** For any  $b \in \Sigma^*$ , there exists at most three  $v \in \mathbb{N}^*$  such that  $E(v) = b$ .

Therefore for any  $b \in \Sigma^*$ , at most three continued fraction cylinders, say  $[v]$ ,  $[v, i]$  and  $[v, i, j]$  get mapped to  $b$ . Therefore we pad two additional bits of information to  $E(v)$ , say  $b_1(v).b_2(v)$  to identify the continued fraction cylinder that  $E(v)$  corresponds to.

► **Definition 17** (One-one binary encoding). For  $v \in \mathbb{N}^*$ , let  $\mathcal{E}(v) = E(v).b_1(v).b_2(v)$ . This forms a one to one binary encoding of  $v$ .

We define Kolmogorov complexity of continued fraction string  $v \in \mathbb{N}^*$  as the Kolmogorov complexity of  $\mathcal{E}(v)$ .

► **Definition 18** (Kolmogorov complexity of continued fraction strings). For any  $v \in \mathbb{N}^*$ , define  $K_{\mathcal{E}}(v) = K(\mathcal{E}(v))$ .

**Notation.** By the invariance theorem of Vishnoi [28], for any  $v \in \Sigma^*$ ,  $K_{\mathcal{E}}$  is at most an additive constant more than the complexity of  $v$  as defined in [28]. Hence, we drop the suffix and denote the above complexity as  $K(v)$ .

In the proof of Theorem 14, Mayordomo [16] provides the construction of an  $s$ -gale that succeeds on all  $X$  for which  $s > s' > \liminf_{n \rightarrow \infty} \frac{K(X \upharpoonright n)}{n}$ . We extend the construction to the setting of continued fractions.

Additionally, we take a convex combination of gales to remove the dependence of the  $s$ -gale on the parameter  $s'$ . Due to this, we obtain the notion of an optimal lower semicomputable continued fraction  $s$ -gale. This notion is crucial in the proofs we use in the upcoming sections.

► **Definition 19.** Given  $0 < s' < s \leq 1$  let

$$G_{s'} = \{w \in \mathbb{N}^* \mid K(w) \leq -s' \log(\mu(w))\}.$$

Consider the following function  $d_{s'} : \mathbb{N}^* \rightarrow [0, \infty)$  defined by

$$d_{s'}(v) = \frac{1}{\gamma^s(v)} \left( \sum_{w \in G_{s'}; v \sqsubseteq w} \gamma^{s'}(w) + \sum_{w \in G_{s'}; w \sqsubset v} \gamma^{s'}(w) \frac{\gamma(v)}{\gamma(w)} \right).$$

Now for each  $i \in \mathbb{N}$ , let  $s_i = s(1 - 2^{-i})$ . Finally, define  $d^* : \mathbb{N}^* \rightarrow [0, \infty)$  by

$$d^*(v) = \sum_{i=1}^{\infty} 2^{-i} d_{s_i}(v).$$

We now go on to show that the function  $d^*$  given in Definition 19 is a lower semicomputable  $s$ -gale. Additionally, it succeeds on all continued fraction sequences  $Y$  for which the Kolmogorov complexity of its prefixes,  $K(Y \upharpoonright n)$  dips below  $s \times -\log(\mu(Y \upharpoonright n))$  infinitely often.

► **Lemma 20.** For any  $0 < s \leq 1$ , there exists a lower semicomputable continued fraction  $s$ -gale  $d^* : \mathbb{N}^* \rightarrow [0, \infty)$  that succeeds on all  $Y \in \mathbb{N}^{\infty}$  such that  $\liminf_{n \rightarrow \infty} \frac{K(Y \upharpoonright n)}{-\log(\mu(Y \upharpoonright n))} < s$ .

We refer to Downey and Hirschfeldt's (Theorem 13.3.4 [5]) proof of the lower bound on constructive dimension using Kolmogorov complexity. The proof fundamentally uses properties of the universal lower semicomputable super-martingale.

For any real having continued fraction dimension less than  $s$ , we obtain a lower semicomputable binary  $s$ -gale that succeeds on it from Lemma 13. We use the success of this binary  $s$ -gale along with the same properties of the universal lower semicomputable super-martingale, to prove the following lemma.

► **Lemma 21.** For any  $Y \in \mathbb{N}^{\infty}$  and any  $s > \text{cdim}_{CF}(Y)$ , we have  $\liminf_{n \rightarrow \infty} \frac{K(Y \upharpoonright n)}{-\log(\mu(Y \upharpoonright n))} \leq s$ .

Therefore, we have the following Kolmogorov complexity based characterization of effective continued fraction dimension.

► **Theorem 22.** For any  $Y \in \mathbb{N}^{\infty}$ ,

$$\text{dim}_{CF}(Y) = \liminf_{n \rightarrow \infty} \frac{K(Y \upharpoonright n)}{-\log(\mu(Y \upharpoonright n))}.$$

**Proof.** For any  $Y \in \mathbb{N}^{\infty}$ , let  $s^* = \liminf_{n \rightarrow \infty} \frac{K(Y \upharpoonright n)}{-\log(\mu(Y \upharpoonright n))}$ .

For any  $s > s^*$ , from Lemma 20, it follows that there exists a lower semicomputable  $s$ -gale  $\mathcal{D}$  that succeeds on  $Y$ . Hence  $\text{dim}_{CF}(Y) \leq s^*$ .

For any  $s > \text{dim}_{CF}(Y)$ , from Lemma 21, we have that  $s^* \leq s$ . Therefore, we have  $s^* \leq \text{dim}_{CF}(Y)$ . ◀

#### 4.1 Optimal gales and effective continued fraction dimension of a set

Lutz [13] utilizes the notion of the optimal constructive subprobability supermeasure  $\mathbf{M}$  on the Cantor space [29] to provide a notion of an optimal constructive supergale.

We note that using Theorem 22, the gale that we obtain from Lemma 20 leads to an analogous notion in the continued fraction setting. We call the continued fraction  $s$ -gale  $d^*$  thus obtained as the *optimal lower semicomputable continued fraction  $s$ -gale*.

► **Lemma 23.** *For any  $s > 0$ , there exists a lower semicomputable continued fraction  $s$ -gale  $d^* : \mathbb{N}^* \rightarrow [0, \infty)$  such that for all  $Y \in \mathbb{N}^\infty$  with  $\text{cdim}_{CF}(Y) < s$ ,  $d^*$  succeeds on  $Y$ .*

**Proof.** For all  $Y \in \mathbb{N}^*$  such that  $\text{cdim}_{CF}(Y) < s$ , from Theorem 22, it follows that  $\liminf_{n \rightarrow \infty} \frac{K(Y \upharpoonright n)}{-\log(\mu(Y \upharpoonright n))} < s$ . Now applying Lemma 20, we see that the given lemma holds. ◀

Lutz (Theorem 4.1 in [13]) shows that the effective dimension of a set is precisely the supremum of effective dimensions of individual elements in the set, that is for all  $X \subseteq [0, 1]$ ,  $\text{cdim}(X) = \sup_{S \in X} \text{cdim}(S)$ . Using the notion of the optimal lower semicomputable continued fraction  $s$ -gale from Lemma 23, we extend this result to continued fraction dimension.

► **Theorem 24.** *For all  $\mathcal{F} \subseteq [0, 1]$ ,  $\text{cdim}_{CF}(\mathcal{F}) = \sup_{Y \in \mathcal{F}} \text{cdim}_{CF}(Y)$ .*

**Proof.** For any  $s > \text{cdim}_{CF}(\mathcal{F})$ , for all  $Y \in \mathcal{F}$  there exists a lower semicomputable continued fraction  $s$ -gale that succeeds on  $Y$ . Thus we have  $\sup_{Y \in \mathcal{F}} \text{cdim}_{CF}(Y) \leq s$ .

Take any any  $s > \sup_{Y \in \mathcal{F}} \text{cdim}_{CF}(Y)$ . It follows that for all  $Y \in \mathcal{F}$ ,  $\text{cdim}_{CF}(Y) < s$ . Therefore from Lemma 23, we have that there exists a lower semicomputable continued fraction  $s$ -gale  $d^* : \mathbb{N}^* \rightarrow [0, \infty)$  that succeeds on all  $Y \in \mathcal{F}$ . Therefore,  $\text{cdim}_{CF}(\mathcal{F}) \leq s$ . ◀

## 5 Reals with unequal Effective Dimension and Effective Continued Fraction Dimension

In this section, we show that for any set of reals  $\mathcal{F} \subseteq [0, 1]$ , the effective Hausdorff effective dimension of  $\mathcal{F}$  cannot exceed its effective continued fraction dimension. We show that this cannot be improved to an equality. Hence, this inequality is strict in general. We show this by proving the existence of a real such that its effective continued fraction dimension is strictly greater its effective dimension.

### 5.1 Effective Hausdorff dimension is at most the effective continued fraction dimension

► **Theorem 25.** *For any  $\mathcal{F} \subseteq [0, 1]$ ,  $\text{cdim}(\mathcal{F}) \leq \text{cdim}_{CF}(\mathcal{F})$ .*

**Proof.** Let  $s > s' > \text{cdim}_{CF}(\mathcal{F})$ . By definition, there exists a lower semicomputable continued fraction  $s'$ -gale  $d : \mathbb{N}^* \rightarrow [0, \infty)$  such that  $\mathcal{F} \subseteq S^\infty[d]$ .

Take any  $Y \in S^\infty[d]$ . Let  $X \in \Sigma^\infty$  be the corresponding binary representation of  $Y$ . By definition, for any  $m \in \mathbb{N}$ , there exists an  $n \in \mathbb{N}$  such that  $d(Y \upharpoonright n) > m$ . Let  $v = Y \upharpoonright n$ .

Using Lemma 7, we get two binary cylinders  $w_1$  and  $w_2$  such that  $C_v \subseteq C_{w_1} \cup C_{w_2}$  such that  $\mu(w_1) = \mu(w_2) \leq 2\mu(v)$ . We have that since  $v \sqsubseteq Y$ ,  $w_1 \sqsubseteq X$  or  $w_2 \sqsubseteq X$ . Without loss of generality assume that  $w_1 \sqsubseteq X$ .

From Lemma 13, we obtain a lower semicomputable  $s$ -gale  $h$  such that  $h(w_1) \geq c_s \cdot d(v) \geq c_s \cdot m$  for some positive constant  $c_s$ .

Since  $m$  is arbitrary, we see that  $h$  succeeds on  $X$ . ◀

## 5.2 Reals with unequal effective Hausdorff and effective continued fraction dimensions

We now provide the main construction of the paper, utilizing the results we show in previous sections.

We first require some technical lemmas about the estimation of Lebesgue measure of a continued fraction cylinder in terms of digits of the continued fraction. Some of the bounds derived in this section may be of independent interest.

In combinatorial arguments, the Gauss measure is often difficult to deal with directly, and it is convenient to use the Lebesgue measure, and derive inequalities.

The following equation, Proposition 1.2.7 in Kraaikamp and Iosifescu [9], is extremely useful in deriving an estimate for the Lebesgue measure of continued fraction cylinders. We derive consequences of this Lemma below, and these are crucial in estimating the dimension of the continued fraction we construct in Section 5. Note that the bounds for Gauss measure are not simple to derive directly.

► **Lemma 26** (Kraaikamp, Iosifescu [9]). *For any  $v = [a_1, \dots, a_n]$  and  $i \in \mathbb{N}$ ,*

$$\frac{\mu([v, i])}{\mu([v])} = \frac{s_n + 1}{(s_n + i)(s_n + i + 1)}$$

where  $s_n = [a_n, \dots, a_1]$  is the rational corresponding to the reverse of string  $v$ .

The lemma given above gives the following bounds on the Lebesgue measure of a continued fraction cylinder in terms of the digits of the continued fraction.

► **Lemma 27.** *For any  $v = [a_1, \dots, a_k] \in \mathbb{N}^k$  we have*

$$\prod_{i=1}^k \frac{1}{(a_i + 1)(a_i + 2)} \leq \mu(v) \leq \prod_{i=1}^k \frac{2}{a_i (a_i + 1)}$$

► **Lemma 28.** *Let  $v = [a_1 \dots a_k] \in \mathbb{N}^*$ . Then for any  $a, b \in \mathbb{N}$  such that  $b > a$ ,*

$$\mu\left(\bigcup_{i=a}^b [v, i]\right) \leq \frac{2}{a} \prod_{i=1}^k \frac{2}{a_i (a_i + 1)}$$

The following lemma is a direct constructive extension of the proof by Lutz [12]. Using this technique, we convert a set of computably enumerable prefix free binary covers into a lower semicomputable binary  $s$ -gale.

► **Lemma 29** (Lutz [12]). *For all  $n \in \mathbb{N}$ , and  $\mathcal{F} \subseteq [0, 1]$ , if there is a computably enumerable prefix free binary cover  $\{B_i^n\}$  of  $\mathcal{F}$ , such that  $\sum_i |B_i^n|^s < 2^{-n}$ , then there exists a lower semicomputable binary  $s$ -gale that succeeds on  $\mathcal{F}$ .*

We now proceed to show the construction of the set  $\mathcal{F}$ . The definition uses a parameter  $\mathbf{s}$ . We later go on to show that for all such  $\mathcal{F}$ , there exists an element  $Y \in \mathcal{F}$  such that the effective continued fraction dimension of  $Y$  is greater than 0.5. We also go on to show that  $\text{cdim}(F) \leq \mathbf{s}$ .

We first provide the stage-wise construction of a set  $\mathcal{F}_k \subseteq [0, 1]$ , such that for each  $k \in \mathbb{N}$   $\mathcal{F}_{k+1} \subseteq \mathcal{F}_k$ . We then define the set  $\mathcal{F}$  using an infinite intersection of the sets  $\mathcal{F}_k$ .

► **Definition 30.** Let  $0 < s < 0.5$ . Define  $a_1 = 1$ . For any  $k \in \mathbb{N}$ , such that  $k > 1$ , define  $a_k$  inductively as:

$$a_k = 2 \left( k \prod_{i=1}^{k-1} 100a_i \right)^{1/s}.$$

For any  $k \in \mathbb{N}$ , define  $b_k = 50 \cdot a_k$ . Take  $\mathcal{F}_0 = \lambda$ .

Let  $\mathcal{F}_k = \{[v_1 \dots v_k] \in \mathbb{N}^k \text{ such that } v_i \in [a_i, b_i] \text{ for } i \in 1 \text{ to } k\}$ . Finally define

$$\mathcal{F} = \bigcap_{k=1}^{\infty} \mathcal{F}_k.$$

We use the bounds obtained from Lemma 26, along with basic properties of harmonic numbers to prove the following property of measures of continued fraction sub cylinders.

► **Lemma 31.** For any  $x \in \mathbb{N}^*$ ,  $s \leq 0.5$  and  $a_k, b_k \in \mathbb{N}$  such that  $b_k = 50 \cdot a_k$ ,

$$\sum_{i=a_k}^{b_k} \gamma^s([x, i]) > c \gamma^s([x]) \text{ for some } c > 1.$$

Using the bound derived above, we show that for  $s = 0.5$ , the optimal  $s$ -gale  $d^*$  formulated in Section 4.1 does not succeed on some sequence in  $Y \in \mathcal{F}$ . Using this we establish that  $\text{cdim}_{CF}(Y) \geq 0.5$ .

► **Lemma 32.** There exists a  $Y \in \mathcal{F}$  such that  $\text{cdim}_{CF}(Y) \geq 0.5$ .

**Proof.** Let  $s = 0.5$ . Consider the continued fraction  $s$ -gale  $d^*$  from Lemma 23. It follows that for all  $Y \in \mathbb{N}^*$ , if  $d^*$  does not succeed on any  $Y \in \mathbb{N}^*$ , then  $\text{cdim}_{CF}(Y) \geq s$ .

Consider any  $v \in \mathbb{N}^*$ , let  $\text{rank}(v) = k$ . From lemma 31, we have that for some  $c > 1$ ,

$$\sum_{i=a_k}^{b_k} \gamma^s([v, i]) > c \cdot \gamma^s([v]).$$

Now if  $\forall i \in [a_k, b_k]$ ,  $d^*([v, i]) \geq \frac{1}{c} \cdot d^*(v)$ , from the  $s$ -gale condition it follows that  $d^*(v) \gamma^s(v) \geq \frac{d^*(v)}{c} \sum_{i=a_k}^{b_k} \gamma^s([v, i]) > d^*(v) \gamma^s(v)$ , which is a contradiction.

Therefore, it follows that for all  $v \in \mathbb{N}^*$ , there exists some  $n_v \in [a_k, b_k]$  such that  $d^*([v, i]) < \frac{1}{c} \cdot d^*(v)$ .

Let  $v_0 = \lambda$ , for each  $i \in \mathbb{N}$ , we define  $v_i = [v_{i-1}, n_{v_{i-1}}]$ . Take  $Y = \bigcap_{i=1}^{\infty} C_{v_i}$ , it follows that  $Y \in \mathcal{F}$ . Taking  $d^*(\lambda) = k$  we get  $d^*(Y \upharpoonright n) < \frac{k}{c^n}$ . Therefore the continued fraction  $s$ -gale  $d^*$  does not succeed on  $Y$ . Hence  $\text{cdim}_{CF}(Y) \geq 0.5$ . ◀

From this, it follows that the constructive dimension of the entire set  $\mathcal{F}$  is also greater than or equal to 0.5.

► **Lemma 33.**  $\text{cdim}_{CF}(\mathcal{F}) \geq 0.5$ .

**Proof.** From Theorem 24, we get that  $\text{cdim}_{CF}(\mathcal{F}) = \sup_{Y \in \mathcal{F}} \text{cdim}_{CF}(Y)$ . From Lemma 32, it follows that there exists a  $Y \in \mathcal{F}$  such that  $\text{cdim}_{CF}(Y) \geq 0.5$ . Combining these two, we get that  $\text{cdim}_{CF}(\mathcal{F}) \geq 0.5$ . ◀

Now we show that the effective Hausdorff dimension of all points in the set  $\mathcal{F}$  is less than  $s$ . Using ideas from [2], we devise a set of covers  $S_k$  for  $\mathcal{F}$ , by combining adjacent continued fraction cylinders into a single cover.

Using the bounds derived on Lebesgue measure of continued fraction cylinders, we show that for the set of covers  $S_k$  for  $\mathcal{F}$ , the  $s$ -dimensional Hausdorff measure shrinks arbitrarily small.

## 70:12 Effective Continued Fraction Dimension

► **Lemma 34.** For  $k \in \mathbb{N}$ , let  $S_k = \{ \bigcup_{i=a_k}^{b_k} [v, i] \text{ for } v \in \mathcal{F}_{k-1} \}$ . Then,  $\sum_{S \in S_k} \mu^{\mathbf{s}}(S) \leq 1/k$ .

**Proof.** The largest element in  $S_k$  is  $I = [(a_1 \dots a_{k-1}, a_k), [(a_1 \dots a_{k-1}, b_k)]$ . The number of elements in  $S_k$  equals  $\prod_{i=1}^{k-1} (b_i - a_i)$ . Additionally, we have  $b_i = 50a_i$  for all  $i \in \mathbb{N}$ . Therefore,

$$\sum_{S \in S_k} \mu^{\mathbf{s}}(S) \leq \mu^{\mathbf{s}}(I) \prod_{i=1}^{k-1} 50a_i.$$

From Lemma 28, it follows that  $\mu(I) \leq \frac{2}{a_k} \prod_{i=1}^{k-1} \frac{2}{a_i(a_i+1)}$ . Therefore,

$$\begin{aligned} \sum_{S \in S_k} \mu^{\mathbf{s}}(S) &\leq \left( \frac{2}{a_k} \prod_{i=1}^{k-1} \frac{2}{a_i^2} \right)^{\mathbf{s}} \prod_{i=1}^{k-1} (50a_i) \\ &\leq \frac{2^{\mathbf{s}}}{a_k^{\mathbf{s}}} \prod_{i=1}^{k-1} 100a_i \end{aligned}$$

Since  $a_k = 2(k \prod_{i=1}^{k-1} 100a_i)^{1/\mathbf{s}}$ , this value is less than  $1/k$ . ◀

To show that the constructive dimension of  $\mathcal{F}$  is less than  $\mathbf{s}$ , we construct a lower semicomputable binary  $\mathbf{s}$ -gale that succeeds on  $\mathcal{F}$ . Using standard techniques, we first convert the covers obtained in Lemma 34 to a set of binary covers of  $\mathcal{F}$ . Finally applying Lemma 29, we convert the binary covers into a semicomputable  $\mathbf{s}$ -gale that succeeds on  $\mathcal{F}$ .

► **Lemma 35.**  $\text{cdim}(\mathcal{F}) \leq \mathbf{s}$ .

**Proof.** Given  $k \in \mathbb{N}$ , from Lemma 34, we have that for  $S_k = \{ \bigcup_{i=a_k}^{b_k} [v, i] \text{ for } v \in \mathcal{F}_{k-1} \}$ ,  $\sum_{S \in S_k} \mu^{\mathbf{s}}(S) \leq 1/k$ . For each  $S \in S_k$ , using Lemma 7, we get that for the two smallest consecutive binary cylinders say  $b_1(S)$  and  $b_2(S)$  that cover  $S$ , we have that  $\mu(b_1) = \mu(b_2) \leq 2\mu(S)$ .

Hence the set  $B_k = \{ \{b_1(S)\} \cup \{b_2(S)\} \text{ such that } S \in S_k \}$  forms a binary cover of  $S_k$ . Also from Lemma 7, we have that  $\sum_{b \in B_k} \mu^{\mathbf{s}}(b) \leq 2^{1+\mathbf{s}} \sum_{S \in S_k} \mu^{\mathbf{s}}(S) \leq 2^{1+\mathbf{s}}/k$ .

Note that the set  $S_k$  is computable as  $a_k$  and  $b_k$  are computable for all  $k$ . Given any interval  $S$ ,  $b_1(S)$  and  $b_2(S)$  are also computable. Hence the set  $B_k$  is computable.

Since  $B_k$  is a finite set, we can remove all  $v \in B_k$  such that  $u \sqsubset v$  for some  $u \in B_k$ , to make  $B_k$  prefix free.

For an  $n \in \mathbb{N}$ , taking  $k = \lceil 2^{1+\mathbf{s}} \cdot 2^n \rceil$ , the set  $B_k$  forms a computably enumerable prefix free binary cover of  $\mathcal{F}$  such that  $\sum_{b \in B_k} \mu^{\mathbf{s}}(b) \leq 2^{-n}$ .

Applying Lemma 29, we get that there exists a lower semicomputable  $\mathbf{s}$ -gale that succeeds on  $\mathcal{F}$ . Hence the lemma holds. ◀

We sum up the results from Lemma 32 and Lemma 35 into the following theorem.

► **Theorem 36.** Given any  $0 < \varepsilon < 0.5$ , there exists a  $Y \in \mathbb{N}^*$  such that  $\text{cdim}_{CF}(Y) \geq 0.5$  and  $\text{cdim}(Y) \leq \varepsilon$ .

**Proof.** Given  $0 < \varepsilon < 0.5$ , taking  $\mathbf{s} = \varepsilon$ , construct the set  $\mathcal{F}$  given in Definition 30.

From Lemma 32, it follows that there exists a  $Y \in \mathcal{F}$  such that  $\text{cdim}_{CF}(Y) \geq 0.5$ .

From Lemma 35, it follows that for all  $X \in \mathcal{F}$ ,  $\text{cdim}(X) \leq \varepsilon$ . Hence  $\text{cdim}(Y) \leq \varepsilon$ . ◀

## References

- 1 S. Albeverio, Ganna Ivanenko, Mykola Lebid, and Grygoriy Torbin. On the Hausdorff dimension faithfulness and the Cantor series expansion. *Methods of Functional Analysis and Topology*, 26(4):298–310, 2020.
- 2 Sergio Albeverio, Yuri Kondratiev, Roman Nikiforov, and Grygoriy Torbin. On new fractal phenomena connected with infinite linear IFS. *Math. Nachr.*, 290(8-9):1163–1176, 2017. doi:10.1002/mana.201500471.
- 3 G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.
- 4 Karma Dajani and Cor Kraaikamp. *Ergodic Theory of Numbers*. The Mathematical Association of America, 2002.
- 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 M. Einsiedler and T. Ward. *Ergodic Theory: with a view towards Number Theory*. Graduate Texts in Mathematics. Springer London, 2010. URL: <https://books.google.co.in/books?id=PiDET2fS7H4C>.
- 7 Kenneth Falconer. *Fractal geometry*. John Wiley & Sons, Inc., Hoboken, NJ, second edition, 2003. Mathematical foundations and applications. doi:10.1002/0470013850.
- 8 John M. Hitchcock and Elvira Mayordomo. Base invariance of feasible dimension. *Inform. Process. Lett.*, 113(14-16):546–551, 2013. doi:10.1016/j.ipl.2013.04.004.
- 9 Marius Iosifescu and Cor Kraaikamp. *Metrical theory of continued fractions*, volume 547 of *Mathematics and its Applications*. Kluwer Academic Publishers, Dordrecht, 2002. doi:10.1007/978-94-015-9940-5.
- 10 L. A. Levin. On the notion of a random sequence. *Soviet Mathematics Doklady*, 14:1413–1416, 1973.
- 11 Jack H. Lutz. Gales and the constructive dimension of individual sequences. In *Automata, languages and programming (Geneva, 2000)*, volume 1853 of *Lecture Notes in Comput. Sci.*, pages 902–913. Springer, Berlin, 2000. doi:10.1007/3-540-45022-X\_76.
- 12 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003. doi:10.1137/S0097539701417723.
- 13 Jack H. Lutz. The dimensions of individual strings and sequences. *Inform. and Comput.*, 187(1):49–79, 2003. doi:10.1016/S0890-5401(03)00187-1.
- 14 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM Journal on Computing*, 38:1080–1112, 2008.
- 15 P. Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.
- 16 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inform. Process. Lett.*, 84(1):1–3, 2002. doi:10.1016/S0020-0190(02)00343-5.
- 17 Elvira Mayordomo. Effective dimension in some general metric spaces. In Benedikt Löwe and Glynn Winskel, editors, *Proceedings 8th International Workshop on Developments in Computational Models, DCM 2012, Cambridge, United Kingdom, 17 June 2012*, volume 143 of *EPTCS*, pages 67–75, 2012. doi:10.4204/EPTCS.143.6.
- 18 Elvira Mayordomo. Effective Hausdorff dimension in general metric spaces. *Theory Comput. Syst.*, 62(7):1620–1636, 2018. doi:10.1007/s00224-018-9848-3.
- 19 S. Nandakumar. An effective ergodic theorem and some applications. In *Proceedings of the 40th Annual Symposium on the Theory of Computing*, pages 39–44, 2008.
- 20 Satyadev Nandakumar and Prateek Vishnoi. On continued fraction randomness and normality. *Inform. and Comput.*, 285(part B):Paper No. 104876, 17, 2022. doi:10.1016/j.ic.2022.104876.
- 21 André Nies. *Computability and randomness*, volume 51. OUP Oxford, 2009.
- 22 Yuval Peres and Gyorgiy Torbin. Continued fractions and dimensional gaps. In preparation.

- 23 Adrian-Maria Scheerer. On the continued fraction expansion of absolutely normal numbers, 2017. [arXiv:1701.07979](https://arxiv.org/abs/1701.07979).
- 24 C. P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit*. Springer-Verlag, Berlin, 1971.
- 25 Ludwig Staiger. The Kolmogorov complexity of real numbers. *Theor. Comput. Sci.*, 284(2):455–466, 2002. doi:10.1016/S0304-3975(01)00102-5.
- 26 Joseph Vandehey. Absolutely abnormal and continued fraction normal numbers. *Bulletin of the Australian Mathematical Society*, 94(2):217–223, 2016. doi:10.1017/S0004972716000101.
- 27 Prateek Vishnoi. *Algorithmic Information Theory & Continued Fractions*. PhD thesis, Indian Institute of Technology Kanpur, 2023.
- 28 Prateek Vishnoi. Normality, randomness and Kolmogorov complexity of continued fractions. In *Theory and Applications of Models of Computation: 17th Annual Conference, TAMC 2022, Tianjin, China, September 16–18, 2022, Proceedings*, pages 382–392. Springer, 2023.
- 29 A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russ. Math. Surv.*, 25(6):83–124, 1970. doi:10.1070/rm1970v025n06ABEH001269.

## A Appendix

### A.1 Proof of Lemma 13

**Proof.** Given an  $s'$ -gale  $d : \mathbb{N}^* \rightarrow [0, \infty)$ , let  $h' = H_d$ , the proportional binary  $s'$ -gale of  $d$  given in Definition 10.

For a  $v \in \mathbb{N}^*$ , consider the smallest  $w_1, w_2 \in \Sigma^*$  such that  $C_v \subseteq C_{w_1} \cup C_{w_2}$ . Also we can see that there exists a  $S \subseteq I(w_1) \cup I(w_2)$  such that  $C_v = \cup_{u \in S} C_u$ .

Therefore from Lemma 8, we get  $h'(w_1) + h'(w_2) \geq d(v) \frac{\gamma^{s'}(v)}{\mu^{s'}(w_1)}$ . From Lemma 7, we get that  $\mu(w_1) = \mu(w_2) \leq 2\mu(v)$ . Also from Lemma 5, we have that  $\gamma(v) \leq (\ln 2)^{-1} \mu(v)$ . Therefore,  $h'(w_1) + h'(w_2) \geq (2 \ln 2)^{-s'} d(v)$ . Now for any  $s > s'$ , we have that  $h'(w_1) + h'(w_2) \geq c_1 d(v)$ , where  $c_1 = 1/(2 \ln 2)^s$ .

Now for any  $s > s'$ , consider the smoothed  $s$ -gale  $h = S_{H_d}$  of the  $s'$ -gale  $H_d$  given in Definition 11.

Let  $|w_1| = n$ , and let  $W_1 = P(P(w_1))$  be the parent cylinder of parent of  $w_1$ . Similarly let  $W_2 = P(P(w_2))$ . We see that for any  $W \in \{W_1, W_2\}$ ,  $h_n(W) \geq 2^{s(n-2)} \frac{h'(w_1) + h'(w_2)}{2} \geq c_2 \cdot 2^{sn} \cdot d(v)$ , where  $c_2 = 2^{-(2s+1)} c_1$ .

Take any any  $b \in \Sigma^*$  such that  $C_b \cap C_v \neq \emptyset$  and  $2 \cdot \mu(v) \geq \mu(b) \geq \frac{1}{16} \mu(v)$ . Since  $\mu(b) \leq 2\mu(v)$  and  $\mu(v) \leq 2\mu(w_1)$ , it follows that for some  $W \in \{W_1, W_2\}$ ,  $W \sqsubseteq b$ . Also since  $\mu(b) \geq \frac{1}{16} \mu(v)$ , we have that,  $\mu(b) \geq \frac{1}{32} \mu(w_1)$ .

Therefore, we have that  $h_n(b) \geq 2^{5(s-1)} h_n(W) \geq c_3 \cdot 2^{sn} \cdot d(v)$ , where  $c_3 = c_2 \cdot 2^{5(s-1)}$ .

Since  $h(b) \geq 2^{-sn} h_n(b)$ , we have that  $h(b) \geq c_3 \cdot d(v)$ . ◀

### A.2 Proof of Lemma 31

**Proof.** From Lemma 26, it follows that

$$\begin{aligned} \sum_{a_n}^{b_k} \frac{\mu^s([x, i])}{\mu^s([x])} &= \sum_{a_n}^{b_k} \left( \frac{s_k + 1}{(s_k + i)(s_k + i + 1)} \right)^s \\ &\geq \sum_{a_k}^{b_k} \left( \frac{1}{(i + 1)(i + 2)} \right)^s \end{aligned}$$

The second inequality follows from the fact that  $s_k \in [0, 1]$ .

Using Lemma 5, we get that

$$\sum_{a_n}^{b_k} \frac{\gamma^s([x, i])}{\gamma^s([x])} \geq \sum_{a_k}^{b_k} \left( \frac{1}{2(i+1)(i+2)} \right)^s$$

Putting  $b_k = 50a_k$  and  $s \leq 0.5$ , we get

$$\begin{aligned} \sum_{a_n}^{b_k} \frac{\gamma^s([x, i])}{\gamma^s([x])} &\geq \frac{1}{2} \sum_{a_k}^{50a_k} \frac{1}{i+2} \\ &= 0.5(H(50a_k + 2) - H(a_k + 1)). \end{aligned}$$

where  $H_n$  is the  $n^{\text{th}}$  Harmonic number. From the fact that  $\ln n \leq H_n \leq \ln n + 1$ , we have

$$\begin{aligned} H(50a_k + 2) - H(a_k + 1) &\geq \ln(50a_k) - \ln(2a_k) - 1 \\ &= \ln(25) - 1. \end{aligned}$$

The lemma holds as  $0.5(\ln 25 - 1)$  is greater than 1. ◀



# On the Expressive Power of Regular Expressions with Backreferences

Taisei Nogami ✉

Waseda University, Tokyo, Japan

Tachio Terauchi ✉ 🏠 

Waseda University, Tokyo, Japan

---

## Abstract

---

A *rewb* is a regular expression extended with a feature called *backreference*. It is broadly known that backreference is a practical extension of regular expressions, and is supported by most modern regular expression engines, such as those in the standard libraries of Java, Python, and more. Meanwhile, *indexed languages* are the languages generated by indexed grammars, a formal grammar class proposed by A.V.Aho. We show that these two models' expressive powers are related in the following way: every language described by a rew b is an indexed language. As the smallest formal grammar class previously known to contain rewbs is the class of context sensitive languages, our result strictly improves the known upper-bound. Moreover, we prove the following two claims: there exists a rew b whose language does not belong to the class of stack languages, which is a proper subclass of indexed languages, and the language described by a rew b without a captured reference is in the class of nonerasing stack languages, which is a proper subclass of stack languages. Finally, we show that the hierarchy investigated in a prior study, which separates the expressive power of rewbs by the notion of nested levels, is within the class of nonerasing stack languages.

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

**Keywords and phrases** Regular expressions, Backreferences, Expressive power

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.71

**Related Version** *Full Version*: <https://arxiv.org/abs/2307.08531> [13]

**Funding** This work was supported by JSPS KAKENHI Grant Numbers JP20H04162, JP20K20625, and JP22H03570.

## 1 Introduction

A *rewb* is a regular expression empowered with a certain extension, called *backreference*, that allows preceding substrings to be used later. It is closer to practical regular expressions than the pure ones, and supported by the standard libraries of most modern programming languages. A typical example of a rew b follows:

► **Example 1.** Let  $\Sigma$  be the alphabet  $\{a, b\}$ . The language  $L(\alpha)$  described by the rew b  $\alpha = ({}_1(a + b)^*)_1 \setminus 1$  is  $\{ww \mid w \in \Sigma^*\}$ . Intuitively,  $\alpha$  first *captures* a preceding string  $w \in L((a + b)^*)$  by  $({}_1)_1$ , and second *references* that  $w$  by following  $\setminus 1$ . Therefore,  $\alpha$  matches  $ww$ . Because this  $L(\alpha)$  is a textbook example of a non-context-free language (and therefore non-regular), the expressive power of rewbs exceeds that of the pure ones.

In 1968, A.V.Aho discovered indexed languages with characterizations by two equivalent models: indexed grammars and (one-way<sup>1</sup> nondeterministic, or 1N) nested stack automata (NSA) [1, 2]. The class of indexed languages is a proper superclass of context free languages (CFL), and a proper subclass of context sensitive languages (CSL) [1].

---

<sup>1</sup> “One-way” means that the input cursor will not move back to left. The antonym is “two-way.”



Berglund and van der Merwe [4], and Câmpeanu et al. [5] have shown that the class of rewbs is incomparable with the class of CFLs and is a proper subclass of CSLs. As the first main contribution of this paper, we prove that the language described by a rewb is an indexed language. Since the class of CSLs was the previously known best upper-bound of rewbs, our result gives a novel and strictly tighter upper-bound.

Meanwhile, there is a class of the languages called stack languages [8, 7]. This class corresponds to the model (1N) stack automata (SA), a restriction of NSA. Hence, it trivially follows that the class of stack languages is a subclass of indexed languages. Actually, this containment is known to be proper [2]. Furthermore, a model called nonerasing stack automata (NESA) has been studied in papers such as [8, 11, 14], and its language class is known to be a proper subclass of stack languages [14].

In this paper, we show that every rewb without a captured reference (that is, one in which no reference  $\backslash i$  appears as a subexpression of an expression of the form  $(\alpha)_j$ ) describes a nonerasing stack language. Given our result, the following question is natural: does every rewb describe a (nonerasing) stack language? We show that the answer is no. Namely, we show a rewb that describes a non-stack language. Finally, Larsen [12] has proposed a notion called *nested levels* of a rewb and showed that they give rise to a concrete increasing hierarchy of expressive powers of rewbs by exhibiting, for each nested level  $i \in \mathbb{N}$ , a language  $L_i$  that is expressible by a rewb at level  $i$  but not at any levels below  $i$ . We show that this hierarchy is within the class of nonerasing stack languages, that is, there exists an NESA  $A_i$  recognizing  $L_i$  for every nested level  $i$ . Below, we summarize the main contributions of the paper.

- (a) Every rewb describes an indexed language. (Section 4, Corollary 16)
- (b) Every rewb without a captured reference describes a nonerasing stack language. (Section 4, Corollary 17)
- (c) There exists a rewb that describes a non-stack language. (Section 5, Theorem 18)
- (d) The hierarchy given by Larsen [12] is within the class of nonerasing stack languages. (Section 6, Theorem 20)

Note that by (b) and (c), it follows that there is a rewb that *needs* capturing of references (Section 5, Corollary 19). See also Figure 2 for a summary of the results.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 defines preliminary notions used in the paper such as the syntax and semantics of rewb, SA, NESA, and NSA. Sections 4, 5, and 6 formally state and prove the paper’s main contributions listed above. Section 7 concludes the paper with a discussion on future work. For space, the proofs are in the full paper [13].

## 2 Related Work

First, we discuss related work on rewbs. There are several variants of the syntax and semantics of rewbs since they first appeared in the seminal work by Aho [3]. A recent study by Berglund and van der Merwe [4] summarizes the variants and the relations between them. In sum, there are two variants of the syntax, whether or not a same label may appear as the index of more than one capture (“may repeat labels”, “no label repetitions”), and two variants of the semantics, whether an unbound reference is interpreted as the empty string or an undefined factor ( $\varepsilon$ -semantics,  $\emptyset$ -semantics). As shown in [4], there is no difference in the expressive powers between these two semantics under the “may repeat labels” syntax (therefore, there are three classes with different expressive powers, namely “no label repetitions” with  $\emptyset$ -semantics, “no label repetitions” with  $\varepsilon$ -semantics, and “may repeat labels”). In this paper, we focus on the “may repeat labels” formalization, which has

the highest expressive power of the three and is often studied in formal language theory. We adopt the  $\varepsilon$ -semantics as the semantics of rewbs. Note that the pioneering formalization of rewbs given by Aho [3] has the equivalent expressive power as this class. The rewbs with “may repeat labels” with  $\varepsilon$ -semantics was recently proposed by Schmid with the notion of ref-words and dereferences [15]. Simultaneously, he proposed a class of automata called *memory automata* (MFA), and showed that its expressive power is equivalent to that of rewbs. Freydenberger and Schmid extended MFA to *MFA with trap-state* [6]. Berglund and van der Merwe [4] showed that the class of Schmid’s rewbs is a proper subclass of CSLs, and is incomparable with the class of CFLs. Note that there is a pumping lemma for the formalization given by C ampeanu et al. [5] but it is known not to work for Schmid’s rewbs. As mentioned above, Larsen introduced the notion of nested levels and showed that increase in the levels increases the expressive powers of rewbs [12].

Next, we discuss related work on the three automata used throughout the paper, namely SA, NESA, and NSA. Ginsburg et al. introduced SA as a mathematical model that is more powerful than pushdown automaton (PDA), and NESA as a restricted version of SA [8]. Hopcroft and Ullman discovered a type of Turing machine corresponding to the class of two-way NESA [11]. Ogden proposed a pumping lemma for stack languages and nonerasing stack languages [14]. Aho proposed NSA with a proof of the fact that (1N) NSA and indexed grammars given by himself in [1] are equivalent in their expressive powers, and recognized PDA and SA as special cases of NSA [2]. Aho also showed that the class of indexed languages is a proper superclass of CFLs, and a proper subclass of CSLs [1]. Hayashi proposed a pumping lemma for indexed languages [9].

### 3 Preliminaries

In this section, we formalize the syntax and the semantics of rewbs following the formalization given in [6]. We begin with the syntax. Let  $\Sigma_\varepsilon = \Sigma \uplus \{\varepsilon\}$  and  $[k] = \{1, 2, \dots, k\}$ , where the symbol  $\uplus$  denotes a disjoint union.

► **Definition 2.** For each natural number  $k \geq 1$ , the set of  $k$ -rewbs over  $\Sigma$ , written  $REWB_k$ , and the mapping  $\text{var} : REWB_k \rightarrow \mathcal{P}([k])$  are defined as follows, where  $a \in \Sigma_\varepsilon$  and  $i \in [k]$ :

$$\begin{aligned} (\alpha, \text{var}(\alpha)) ::= & (a, \emptyset) \mid (\backslash i, \{i\}) \mid (\alpha_0 \alpha_1, \text{var}(\alpha_0) \cup \text{var}(\alpha_1)) \mid (\alpha_0 + \alpha_1, \text{var}(\alpha_0) \cup \text{var}(\alpha_1)) \\ & \mid (\alpha_0^*, \text{var}(\alpha_0)) \mid ((\alpha_0)_j, \text{var}(\alpha_0) \uplus \{j\}) \text{ where } j \in [k] \setminus \text{var}(\alpha_0). \end{aligned}$$

We also write  $REWB_0$  for the set REG of regular expressions over  $\Sigma$ , and  $REWB$  for the set of all rewbs, namely  $\bigcup_{k \geq 0} REWB_k$ .

► **Example 3.** For example,  $\varepsilon$ ,  $a$ ,  $\backslash 1$ ,  $a^* \backslash 1$ ,  $(1a^*)_1$ ,  $((1a^*)_1)^*$ ,  $(2a^*)_2 \backslash 2$ ,  $(1a^*)_1 (2b^*)_2 (\backslash 1 + \backslash 2)$ ,  $(2(1(a+b)^*)_1 \backslash 1)_2 \backslash 2 (2 \backslash 1)_2^*$ ,  $((1 \backslash 4 a)_1 (2 \backslash 3)_2 (3 \backslash 2 a)_3 (4 \backslash 1 \backslash 3)_4)^*$  are rewbs. On the other hand,  $(1(1a^*)_1)_1$ ,  $(1a^* \backslash 1)_1$ ,  $(1(2(1a^*)_1)_2)_1$  are not rewbs.

Note that this syntax allows multiple occurrences of captures with the same label, that is, we adopt the “may repeat labels” convention. Next, we define the semantics.

► **Definition 4.** Let  $B_k = \{[i, ]_i \mid i \in [k]\}$ . The mapping  $\mathcal{R}_k : REWB_k \rightarrow \mathcal{P}((\Sigma \uplus B_k \uplus [k])^*)$  is defined as follows, where  $a \in \Sigma_\varepsilon$  and  $i \in [k]$ :

$$\begin{aligned} \mathcal{R}_k(a) &= \{a\}, \quad \mathcal{R}_k(\backslash i) = \{i\}, \quad \mathcal{R}_k(\alpha_0 \alpha_1) = \mathcal{R}_k(\alpha_0) \mathcal{R}_k(\alpha_1), \\ \mathcal{R}_k(\alpha_0 + \alpha_1) &= \mathcal{R}_k(\alpha_0) \cup \mathcal{R}_k(\alpha_1), \quad \mathcal{R}_k(\alpha^*) = \mathcal{R}_k(\alpha)^*, \quad \mathcal{R}_k((\alpha)_i) = \{[i] \mathcal{R}_k(\alpha) [i]\}. \end{aligned}$$

We let  $\Sigma_k^{[*]}$  denote  $\bigcup_{\alpha \in REWB_k} \mathcal{R}_k(\alpha)$ .

► **Example 5.**  $\mathcal{R}_k((1(a+b)^*)_1 \setminus 1) = \{[1] \{a, b\}^* [1] \{1\}\} = \{[1 w]_1 1 \mid w \in \{a, b\}^*\}$ .

That is, we first regard a rewb  $\alpha$  over  $\Sigma$  as a regular expression over  $\Sigma \uplus B_k \uplus [k]$ , deducing the language  $\mathcal{R}_k(\alpha)$ . The second step, described next, is to apply the *dereferencing (partial) function*  $\mathcal{D}_k : (\Sigma \uplus B_k \uplus [k])^* \rightarrow \Sigma^*$  to each of its element.

We give an intuitive description of  $\mathcal{D}_k$ . First,  $\mathcal{D}_k$  scans its input string from the beginning toward the end, seeking  $i \in [k]$ . If such  $i$  is found,  $\mathcal{D}_k$  replaces this  $i$  with the substring obtained by removing the brackets in  $v$  that comes from the preceding  $[i v]_i$  if  $[i$  exists (if this  $[i$  has no corresponding  $]_i$ ,  $\mathcal{D}_k$  becomes undefined). Otherwise,  $\mathcal{D}_k$  replaces this  $i$  with  $\varepsilon$ . The dereferencing function  $\mathcal{D}_k$  repeats this procedure until all elements of  $[k]$  appearing in the string are exhausted, then removes all remaining brackets. We let  $v_{[r]}$  denote the string which  $\mathcal{D}_k$  scans at the  $r^{\text{th}}$  number  $n_r \in [k]$  at the  $r^{\text{th}}$  loop (see the full version [13] for the formal definitions of  $\mathcal{D}_k$  and  $v_{[r]}$ ).

1.  $[1a [2b]_2 2]_1 1$ . In this example,  $\mathcal{D}_k$  encounters  $n_1 = 2$  first, and this 2 corresponds the preceding  $[2b]_2$ , therefore this 2 is replaced with  $v_{[1]} = b$ . As a result, the input string becomes  $[1a [2b]_2 b]_1 1$ .  $\mathcal{D}_k$  repeats this process again. Now,  $\mathcal{D}_k$  locates  $n_2 = 1$  corresponding the preceding  $[1a [2b]_2 b]_1$ , so this 1 is replaced with  $v_{[2]} = a[2b]_2 b$  but with the brackets erased. Therefore we gain  $[1a [2b]_2 b]_1 abb$ . Finally,  $\mathcal{D}_k$  removes all remaining brackets and produces  $abbabb$ . Here is the diagram:  $[1a [2b]_2 2]_1 1 \rightarrow [1a [2b]_2 b]_1 1 \rightarrow [1a [2b]_2 b]_1 abb \rightarrow abbabb$ .
2.  $[1a]_1 1 [1bb]_1 1$ . In this example,  $n_1 = n_2 = 1$ ,  $v_{[1]} = a$ ,  $v_{[2]} = bb$ , and

$$[1a]_1 1 [1bb]_1 1 \rightarrow [1a]_1 a [1bb]_1 1 \rightarrow [1a]_1 a [1bb]_1 bb \rightarrow aabbbb.$$

3.  $abc12$ . In this example,  $n_1 = n_2 = 1$ ,  $v_{[1]} = v_{[2]} = \varepsilon$ , and  $abc12 \rightarrow abc2 \rightarrow abc$ .

Note that an unbound reference is replaced with the empty string  $\varepsilon$ , that is, we adopt the  $\varepsilon$ -semantics. However, as mentioned in Section 2, this semantics' expressive power is equivalent to that of the  $\emptyset$ -semantics under the “may repeat labels” convention (see [4] for the proof). We define the language  $L(\alpha)$  denoted by a  $k$ -rewb  $\alpha \in REWB_k$  to be  $\mathcal{D}_k(\mathcal{R}_k(\alpha)) = \{\mathcal{D}_k(v) \mid v \in \mathcal{R}_k(\alpha)\}$  (Lemmas 6 and 8 ensure that  $L(\alpha)$  is well-defined).

Let  $g : (\Sigma \uplus B_k)^* \rightarrow \Sigma^*$  denote the free monoid homomorphism where  $g(x)$  is  $x$  for each  $x \in \Sigma$ , and  $\varepsilon$  for each  $x \in B_k$ . Every  $v \in (\Sigma \uplus B_k \uplus [k])^*$  can be written uniquely in the form  $v = v_0 n_1 v_1 \cdots n_m v_m$ , where  $m \geq 0$  (denoted by  $\text{cnt } v$ ), and  $v_r \in (\Sigma \uplus B_k)^*$  and  $n_r \in [k]$  for each  $r \in \{0, \dots, m\}$ . Here, let  $y_0 \triangleq v_0$  and for each  $r \in \{1, \dots, m\}$ ,  $y_r \triangleq v_0 n_1 v_1 \cdots n_r v_r$ . A string  $v = v_0 n_1 v_1 \cdots n_m v_m$  over  $\Sigma \uplus B_k \uplus [k]$  is said to be *matching* if

$$\forall r \in \{1, \dots, m\}. \forall x_1, x_2. y_{r-1} = x_1 [n_r x_2 \implies (\exists x'_2, x_3. x_2 = x'_2]_{n_r} x_3 \wedge x'_2 \notin ]_{n_r}]_{n_r}$$

holds. Intuitively, a string  $v$  being matching means that for all  $n_r \in [k]$  in  $v$ , if there exists a left bracket  $[n_r$  in the string immediately before  $n_r$ , then there is a right bracket  $]_{n_r}$  in between this  $[n_r$  and  $n_r$ . The following three lemmas follow.

► **Lemma 6.** *Given a matching string  $v$ ,  $\mathcal{D}_k(v) = g(v_0) g(v_{[1]}) g(v_1) \cdots g(v_{[m]}) g(v_m)$ .*

► **Lemma 7.** *A prefix of a matching string is matching. That is, if we decompose a string  $v$  into  $v = xy$ ,  $x$  is matching. Moreover,  $x_{[r]} = v_{[r]}$  holds for each  $r = 1, \dots, \text{cnt } x (\leq \text{cnt } v)$ .*

► **Lemma 8.** *Every  $v \in \Sigma_k^{[*]}$  is matching.*

Next, we recall the notions of SA, NESAs, and NSAs. In this paper, we unify their definitions based on [2, 7] to clarify the different capabilities of these models. First, we review NFA. Here is the definition in the textbook by Hopcroft et al. [10]:

► **Definition 9** ([10], p.57). A *nondeterministic finite automaton*  $N$  is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  a finite set of input symbols (also called alphabet),  $q_0 \in Q$  a start state,  $F \subseteq Q$  a set of final states, and  $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  a transition function.

As well known, the transition function  $\delta$  can be *extended* to  $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$  where  $\hat{\delta}(q, w)$  represents the set of all states reachable from  $q$  via  $w$ . Let  $q \xrightarrow{a}_N q'$  denote  $q' \in \delta(q, a)$ , and  $q \xrightarrow{w}_N q'$  denote  $q' \in \hat{\delta}(q, w)$ . With this notation, the language of an NFA  $N$  can be written as follows:  $L(N) = \left\{ w \in \Sigma^* \mid \exists q_f \in F. q_0 \xrightarrow{w}_N q_f \right\}$ .

A pushdown automaton (PDA) is an NFA equipped with a *stack* such that the PDA may write and read its *stack top* with a transition. A *stack automaton* (SA) is “an extended PDA”, which can reference not only the top but inner content of the stack. That is, while the *stack pointer* of a PDA is fixed to the top, an SA allows its pointer to move left and right and read a stack symbol pointed to by the pointer. However, the only place on the stack that can be rewritten is the top, as in PDA. Formally, a (1N) SA  $A$  is a 9-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, \$, F)$  satisfying the following conditions: the components  $Q, \Sigma, q_0$  and  $F$  are the same as those of NFA.  $\Gamma (\neq \emptyset)$  is a finite set of stack symbols, and  $Z_0 \in \Gamma$  is an initial stack symbol. The stack symbol  $\# \notin \Sigma \cup \Gamma$  (resp.  $\$ \notin \Sigma \cup \Gamma$ ) is always and only written at the leftmost (bottom) (resp. the right most (top)) of the stack.<sup>2</sup> The transition function  $\delta$  has the following two modes, where  $L, S, R \notin (\Sigma \cup \Gamma) \uplus \{\#, \$\}$ ,  $\Delta_i \triangleq \{S, R\}$ , and  $\Delta_s \triangleq \{L, S, R\}$ :

- (i) (pushdown mode)  $Q \times \Sigma \times \Gamma\$ \rightarrow \mathcal{P}(Q \times \Delta_i \times \Gamma*\$)$ ,
- (ii) (stack reading mode) (a)  $Q \times \Sigma \times \Gamma\$ \rightarrow \mathcal{P}(Q \times \Delta_i \times \{L\})$ , (b)  $Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Delta_i \times \Delta_s)$ ,  
(c)  $Q \times \Sigma \times \{\#\} \rightarrow \mathcal{P}(Q \times \Delta_i \times \{R\})$ .

Intuitively,  $\delta$  works as follows (Definition 10 provides the formal semantics). (i) The statement  $(q', d, w\$) \in \delta(q, a, Z\$)$  says that whenever the current state is  $q$ , the input symbol is  $a$ , and the pointer references the top symbol  $Z$ , the machine can move to the state  $q'$ , move the input cursor along  $d$ , and replace  $Z$  with the string  $w$ . (ii) The statement (b)  $(q', d, e) \in \delta(q, a, Z)$  says that whenever the current state is  $q$ , the input symbol is  $a$ , and the pointer references the symbol  $Z$ , the machine can move to the state  $q'$ , move the input cursor along  $d$ , and move the pointer along  $e$ . The statements (a) and (c) are similar to (b) except that the direction in which the pointer can move is restricted lest the pointer go out of the stack. In particular, an SA that cannot erase a symbol once written on the stack is called a *nonerasing stack automaton* (NESAs). That is, a (1N) nonerasing stack automaton is an SA whose transition function  $\delta$  satisfies the condition that, in (i) (pushdown mode),  $(q', d, w\$) \in \delta(q, a, Z\$)$  implies  $w \in Z\Gamma^*$ . To formally describe how SA works, we define a tuple called *instantaneous description* (ID), which consists of a state, an input string, and a string representation of the stack, and define the binary relation  $\vdash_A$  over the set of these tuples. Let  $\bar{L} = -1$ ,  $\bar{S} = 0$ , and  $\bar{R} = 1$ .

► **Definition 10.** Let  $A$  be an SA  $(Q, \Sigma, \Gamma, \delta, q_0, \#, \$, F)$ . An element of the set  $I = Q \times \Sigma^* \times \{\#\}(\Gamma \uplus \{1\})^* \{\$\}$  is called *instantaneous description*, where the stack symbol  $1 \notin \Gamma$  stands for the position of stack pointer. Moreover, let  $\vdash_A$  (or  $\vdash$  when  $A$  is clear) be the smallest binary relation over  $I$  satisfying the following conditions:

<sup>2</sup> These special symbols  $\#, \$$  representing “bottom” and “top” of the stack respectively do not appear in [7] and are introduced anew in this paper to define NESAs and NSAs, which will be defined later, in the style of [2]. In fact, SA defined in [7] is not capable of directly discerning whether the stack pointer is at the top or not. Although it is not difficult to see that directly adding the ability does not increase the expressive power of SA, the ability is directly in NESAs as seen in [11, 14]. Therefore, to make it easy to see that NESAs is a restriction of SA, we define SA to also directly have the ability.

- (i)  $(q, a_i \cdots a_k, \#yZ \uparrow \$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#yw \uparrow \$)$  if  $(q', d, w\$) \in \delta(q, a_i, Z\$)$ .<sup>3</sup>
- (ii) (a)  $(q, a_i \cdots a_k, \#yZ \uparrow \$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#y \uparrow Z\$)$  if  $(q', d, L) \in \delta(q, a_i, Z\$)$ .  
 (b) if  $(q', d, e) \in \delta(q, a_i, Z)$  and  $Z = Z_j$ ,  $1 \leq j < n$ , then  
 $(q, a_i \cdots a_k, \#Z_1 \cdots Z_j \uparrow \cdots Z_n \$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#Z_1 \cdots Z_{j+\bar{e}} \uparrow \cdots Z_n \$)$ .  
 (c)  $(q, a_i \cdots a_k, \# \uparrow Zy \$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#Z \uparrow y \$)$  if  $(q', d, R) \in \delta(q, a_i, \#)$ .

Note that  $L \notin \Delta_i$ , which means the input cursor will not move back to left. We say that  $A$  accepts  $w \in \Sigma^*$  if there exist  $y_1, y_2 \in \Gamma^*$ , and  $q_f \in F$  such that  $(q_0, w, \#Z_0 \uparrow \$) \vdash_A^* (q_f, \varepsilon, \#y_1 \uparrow y_2 \$)$ . Let  $L(A)$  denote the set of all strings accepted by  $A$ .

We next define *nested stack automaton* (NSA) which is SA extended with the capability to create and remove substacks. For instance, suppose that the stack is  $\#a_1a_2 \uparrow a_3 \$$  and we are to create a new substack containing  $b_1b_2$ :

$$\#a_1 \underline{c b_1 b_2} \uparrow \$ a_2 a_3 \$ \tag{1}$$

Note that the new substack  $c b_1 b_2 \$$  is embedded below the symbol  $a_2$  indicated by the stack pointer, and the pointer moves to the top of the created substack. The creation of the inner substack narrows the range within which the stack pointer can move as indicated by the underlined part  $\#a_1 \underline{c b_1 b_2} \uparrow \$$ . While the bottom of the entire stack is always fixed by the leftmost symbol  $\#$ , the top of the embedded substack is regarded as the top of the entire stack. The inner substacks are allowed to be embedded endlessly and everywhere, whereas the writing in the pushdown mode is still restricted to the top of the stack:

$$\#a_1 \underline{c b_1 b_2} \uparrow \$ a_2 a_3 \$ \xrightarrow{L} \#a_1 \underline{c b_1} \uparrow b_2 \$ a_2 a_3 \$ \xrightarrow{\text{create}} \#a_1 \underline{c c c_1 c_2} \uparrow \$ b_1 b_2 \$ a_2 a_3 \$, \tag{2}$$

$$\#a_1 \underline{c} \uparrow b_1 b_2 \$ a_2 a_3 \$ \xrightarrow{L} \#a_1 \uparrow \underline{c b_1 b_2} \$ a_2 a_3 \$ \xrightarrow{\text{create}} \# \underline{c c_1 c_2} \uparrow \$ a_1 \underline{c b_1 b_2} \$ a_2 a_3 \$ \tag{3}$$

We must empty the inner substack and then remove itself in advance whenever we want to reference the right side of the inner substack such as  $a_2, a_3$ . For example, let us empty the inner substack by popping twice from (1) and then removing it:

$$\#a_1 \underline{c b_1 b_2} \uparrow \$ a_2 a_3 \$ \xrightarrow{\text{pop}} \#a_1 \underline{c b_1} \uparrow \$ a_2 a_3 \$ \xrightarrow{\text{pop}} \#a_1 \underline{c} \uparrow \$ a_2 a_3 \$ \xrightarrow{\text{destruct}} \#a_1 a_2 \uparrow a_3 \$ \tag{4}$$

Notice that the stack pointer moves to the right after removing the inner substack. We now define NSA formally. A (1N) nested stack automaton  $A$  is a 10-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, c, \$, F)$  satisfying the following conditions: the components  $Q, \Sigma, \Gamma, q_0, Z_0, \#, \$$  and  $F$  are the same as those of SA. The stack symbol  $c \notin \Sigma \cup \Gamma$  represents the bottom of a substack.<sup>4</sup> The transition function  $\delta$  has the following four modes, where  $\Gamma' \triangleq \Gamma \uplus \{c\}$ :

- (i) (pushdown mode)  $Q \times \Sigma \times \Gamma \$ \rightarrow \mathcal{P}(Q \times \Delta_i \times \Gamma^* \$)$ .
- (ii) (stack reading mode) (a)  $Q \times \Sigma \times \Gamma' \$ \rightarrow \mathcal{P}(Q \times \Delta_i \times \{L\})$ , (b)  $Q \times \Sigma \times \Gamma' \rightarrow \mathcal{P}(Q \times \Delta_i \times \Delta_s)$ ,  
 (c)  $Q \times \Sigma \times \{\#\} \rightarrow \mathcal{P}(Q \times \Delta_i \times \{R\})$ .
- (iii) (stack creation mode)  $Q \times \Sigma \times (\Gamma' \uplus \Gamma' \$) \rightarrow \mathcal{P}(Q \times \Delta_i \times \{c\} \Gamma^* \$)$ .
- (iv) (stack destruction mode)  $Q \times \Sigma \times \{c\} \$ \rightarrow \mathcal{P}(Q \times \Delta_i)$ .

Moreover, we define how NSA works with ID and  $\vdash$  in the same manner as SA. Given an NSA  $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, c, \$, F)$ , we define ID,  $\vdash_A$ , and  $L(A)$  in the same way as Definition 10 (however, we let  $I$  be  $Q \times \Sigma^* \times \{\#\} (\Gamma \uplus \{c, \$, \uparrow\})^* \{\$\}$ ). Here, we only give the rules corresponding to (iii) and (iv) in the definition of  $\delta$  (the others are essentially the same as those of SA):

<sup>3</sup> We regard  $a_{k+1} \cdots a_k$  as  $\varepsilon$ .

<sup>4</sup> Note that the bottom of the entire stack is always represented by  $\#$  and not  $c$ , as mentioned above.

- (iii) if  $(q', d, cw\$) \in \delta(q, a_i, Z)$  and  $Z = Z_j$ ,  $1 \leq j < n$ , then  
 $(q, a_i \cdots a_k, \#Z_1 \cdots Z_j \uparrow \cdots Z_n\$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#Z_1 \cdots cw \uparrow \$Z_j \cdots Z_n\$)$ ,  
and  $(q, a_i \cdots a_k, \#yZ \uparrow \$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#y cw \uparrow \$Z\$)$  if  $(q', d, cw\$) \in \delta(q, a_i, Z\$)$ .
- (iv)  $(q, a_i \cdots a_k, \#y_1 c \uparrow \$Zy_2\$) \vdash_A (q', a_{i+\bar{d}} \cdots a_k, \#y_1 Z \uparrow y_2\$)$  if  $(q', d) \in \delta(q, a_i, c\$)$ .

#### 4 Every rewb describes an indexed language

As described above, to obtain the language  $L(\alpha)$  described by a  $k$ -rewb  $\alpha$ , we derive the regular language  $\mathcal{R}_k(\alpha)$  over the alphabet  $\Sigma \uplus B_k \uplus [k]$  first, then apply the dereferencing function  $\mathcal{D}_k$  to every element of  $\mathcal{R}_k(\alpha)$ . Using this observation, we construct an NSA  $A_\alpha$  recognizing the language  $L(\alpha)$  as follows.

The NSA  $A_\alpha$  is based on an NFA  $N$  recognizing the language  $\mathcal{R}_k(\alpha)$ , in the sense that each transition in  $A_\alpha$  comes from a corresponding transition of  $N$ . The NFA  $N$  has the alphabet  $\Sigma \uplus B_k \uplus [k]$ , and so handles three types of characters. For each transition  $q \xrightarrow[N]{a} q'$  with  $a \in \Sigma$ , i.e., moving from  $q$  to  $q'$  by an input symbol  $a$ ,  $A_\alpha$  also has the same transition except pushing  $a$  to the stack, denoted by  $q \xrightarrow{a/\$ \rightarrow a\$} q'$ . For each transition  $q \xrightarrow[N]{b} q'$  with  $b \in B_k$ , i.e., moving by a bracket  $b$ ,  $A_\alpha$  has the transition pushing  $b$  without consuming input symbols, denoted by  $q \xrightarrow{\varepsilon/\$ \rightarrow b\$} q'$ .<sup>5</sup> For each transition  $q \xrightarrow[N]{i} q'$  with  $i \in [k]$ ,  $A_\alpha$  has a large “transition” that consists of several transitions. In this “transition,”  $A_\alpha$  first seeks the left bracket  $[_i$  of the bracketed string  $[_i v]_i$  within the stack, and checks if the input from the cursor position matches  $v$  character by character while consuming the input, and finally moves to  $q'$  if all characters of  $v$  matched.

A difficult yet interesting point is that NSA cannot check  $v$  against the stack and push  $v$  onto the stack at the same time, that is, after checking a character  $c$  of  $v$ , if  $A_\alpha$  wants to push  $c$  to the stack,  $A_\alpha$  must leave from  $v$ , climb up the stack toward the top, and write  $c$ . However, after the push,  $A_\alpha$  becomes lost by not knowing where to go back to. How about marking the place where  $A_\alpha$  should return in advance? Unfortunately, that does not work; NSA can insert such marks anywhere by creating substacks, but due to the restriction of NSA, it cannot go above the position of the mark, much less climb up to the top. Therefore, NSA cannot directly push the result of a dereference onto the stack.

We cope with this problem as follows. We allow  $j \in [k]$  to appear in  $v$ , and for each appearance of  $j$  in the checking of  $v$ ,  $A_\alpha$  pauses the checking and puts a substack containing the current state as a marker at the stack pointer position. Then,  $A_\alpha$  searches down the stack for the corresponding bracketed string  $]_j v']_j$ , and begins checking  $v'$  if it is found. By repeating this process,  $A_\alpha$  eventually reaches a string  $v'' \in (\Sigma \uplus B_k)^*$  containing no characters of  $[k]$ . Once done with the check of  $v''$ ,  $A_\alpha$  climbs up toward the stack top, finds a marker  $p$  denoting the state to return to, and resumes from  $p$  after deleting the substack containing the marker. By repeating this, if  $A_\alpha$  returns to the position where it initially found  $j$ , it has successfully consumed the substring of the input string corresponding to the dereference of  $j$ . The following lemma is immediate.

► **Lemma 11.** *Let  $k \geq 1$  and  $\alpha \in \text{REWB}_k$ . There exists an NFA  $(Q, \Sigma \uplus B_k \uplus [k], \delta, q_0, F)$  over  $\Sigma \uplus B_k \uplus [k]$  recognizing  $\mathcal{R}_k(\alpha)$  all of whose states can reach some final state, that is,  $\forall q \in Q. \exists w \in (\Sigma \uplus B_k \uplus [k])^*. \exists q_f \in F. q \xrightarrow[N]{w} q_f$ .*

<sup>5</sup> Strictly speaking, our NFA (cf. Definition 9) does not allow consuming the empty string  $\varepsilon$ . However, we can realize the transition  $q \xrightarrow{\varepsilon/\$ \rightarrow b\$} q'$  alternatively by adding  $q \xrightarrow{c/\$ \rightarrow b\$,S} q'$  for each  $c \in \Sigma$ , i.e., moving by  $c$  with the input cursor fixed.

► **Corollary 12.** *Let  $N$  be the NFA in Lemma 11. For all  $q \in Q$  and for all  $w \in (\Sigma \uplus B_k \uplus [k])^*$ , if  $q_0 \xrightarrow{w}_N q$  then  $w$  is matching (see the full version [13] for the proof).*

We show the main theorem (the proof sketch is coming later):

► **Theorem 13.** *For every rewb  $\alpha \in REWB$ , there exists an NSA that recognizes  $L(\alpha)$ .*

The claim obviously holds when  $\alpha$  is a pure regular expression (i.e.,  $\alpha \in REWB_0$ ). Suppose that  $\alpha \in REWB_k$  with  $k \geq 1$ . By Lemma 11, there is an NFA  $N = (Q_N, \Sigma \uplus B_k \uplus [k], \delta_N, q_0, F)$  that recognizes  $\mathcal{R}_k(\alpha)$  and satisfies Corollary 12. We construct an NSA  $A_\alpha = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, \mathfrak{c}, \mathfrak{S}, F)$  as follows. Let  $Q \triangleq Q_N \uplus \{c_i, e_i, r_i \mid i \in [k]\} \uplus \{W_q \mid q \in Q_N\} \uplus \{E_{p,i}, L_{p,i} \mid p \in Q_N \uplus \{e_i \mid i \in [k]\}, i \in [k]\}$ ,  $\Gamma \triangleq \Sigma \uplus B_k \uplus [k] \uplus Q \uplus \{Z_0\}$ , and let  $\delta$  be the smallest relation that, for all  $a \in \Sigma$ ,  $b \in B_k$ ,  $c \in \Sigma$ ,  $i, j \in [k]$ ,  $q, q' \in Q_N$ ,  $Z \in \Gamma$  and  $p \in Q_N \uplus \{e_i \mid i \in [k]\}$ , satisfies the following conditions:

- |  |  |
|--|--|
| (1) $\delta_N(q, a) \ni q' \implies \delta(q, a, Z\mathfrak{S}) \ni (q', R, Za\mathfrak{S})$     | (10) $\delta(e_i, c, [j]) = \{(e_i, S, R)\}$ where $i \neq j$  |
| (2) $\delta_N(q, b) \ni q' \implies \delta(q, c, Z\mathfrak{S}) \ni (q', S, Zb\mathfrak{S})$     | (11) $\delta(e_i, c, [j]) = \begin{cases} \{(r_i, S, R)\} & (i = j) \\ \{(e_i, S, R)\} & (i \neq j) \end{cases}$ |
| (3) $\delta_N(q, i) \ni q' \implies \delta(q, c, Z\mathfrak{S}) \ni (W_{q'}, S, Zi\mathfrak{S})$ | (12) $\delta(e_i, c, j) = \{(c_j, S, \mathfrak{c}e_i\mathfrak{S})\}$ where $i \neq j$                            |
| (4) $\delta(W_q, c, i\mathfrak{S}) = \{(c_i, S, \mathfrak{c}q\mathfrak{S})\}$                    | (13) $\delta(r_i, c, Z) = \{(r_i, S, R)\}$   |
| (5) $\delta(c_i, c, p\mathfrak{S}) = \{(c_i, S, L)\}$  | (14) $\delta(r_i, c, p\mathfrak{S}) = \{(E_{p,i}, S, \mathfrak{S})\}$  |
| (6) $\delta(c_i, c, Z) = \{(c_i, S, L)\}$ where $Z \neq [i, Z_0]$                                | (15) $\delta(E_{p,i}, c, \mathfrak{c}\mathfrak{S}) = \{(L_{p,i}, S)\}$   |
| (7) $\delta(c_i, c, Z_0) = \{(r_i, S, R)\}$  | (16) $\delta(L_{e_j,i}, c, i) = \{(e_j, S, R)\}$   |
| (8) $\delta(c_i, c, [i]) = \{(e_i, S, R)\}$  | (17) $\delta(L_{q,i}, c, i\mathfrak{S}) = \{(q, S, S)\}$   |
| (9) $\delta(e_i, a, a) = \{(e_i, R, R)\}$  |  |

Rule (1) translates  $q \xrightarrow{a}_N q'$  into  $q \xrightarrow{a/\mathfrak{S} \rightarrow a\mathfrak{S}} q'$ , (2) translates  $q \xrightarrow{b}_N q'$  into  $q \xrightarrow{\varepsilon/\mathfrak{S} \rightarrow b\mathfrak{S}} q'$ , and rules (3)–(17) translate  $q \xrightarrow{i}_N q'$  into a large “transition” to consume the string that corresponds to the dereference of  $i$ . The details of the “transition” are as follows. By looking at the underlying  $N$  with rule (3),  $A_\alpha$  finds a state  $q'$  that it should go back to after going throughout the “transition,” and goes to the state  $W_{q'}$  by pushing  $i$  to the stack. At  $W_{q'}$ , by rule (4),  $A_\alpha$  inserts  $\mathfrak{c}q'\mathfrak{S}$  just below  $i$ , and goes to the state  $c_i$ . The state  $c_i$  represents the *call mode* in which  $A_\alpha$  looks for the left-nearest  $[i$  by rules (5) and (6) and proceeds to the state  $e_i$  (*execution mode*) by (8) if it finds  $[i$ . Otherwise (i.e., the case when  $A_\alpha$  arrives at the bottom of the stack), it proceeds to the state  $r_i$  (*return mode*) by rule (7). At  $e_i$ ,  $A_\alpha$  consumes input symbols by checking them against the symbols on the stack (rules (9)–(12)). In particular, rule (9) handles the case when the symbols match. Rules (10) and (11) handle the cases when brackets are read from the stack. The first case of (11) handles the case when the right bracket  $]_i$  is read, and the rules handle the other brackets (i.e.,  $[_j$  or  $]_j$  with  $i \neq j$ ) by simply skipping them (note that  $]_j = [i$  cannot happen since we started from the left-nearest  $[i$ ). Reading  $j \in [k]$ , by rule (12),  $A_\alpha$  inserts  $\mathfrak{c}e_i\mathfrak{S}$  just below  $j$  and goes to  $c_j$  to locate the corresponding  $]_j$  (here,  $j \neq i$  holds by the definition of the syntax). At  $r_i$ ,  $A_\alpha$  proceeds to return to the state  $p$  that passed the control to  $c_i$  (rules (13)–(17)). Since this  $p$  was pushed at the stack top,  $A_\alpha$  first climbs up to the stack top by rule (13), transits to the state  $E_{p,i}$  popping  $p$  by (14), then goes to  $L_{p,i}$  removing the embedded substack by (15), and finally goes back to  $p$  by (16) and (17). A subtle point in the last step is that where the stack pointer should be placed depends on whether  $p$  is a state  $e_j$  (for some  $j \in [k]$ ) or in  $Q_N$ . In the former case, after (15) removes the embedded substack  $\mathfrak{c}e_j\mathfrak{S}$  that was created just below the call to  $i$ , the stack pointer points to  $i$ . However, the stack pointer should shift one more to the right, lest  $A_\alpha$  begins to repeat the call reading  $i$  again by (12). Therefore, (16) correctly handles the case by doing the shift. In the latter case, as stipulated by (17), the stack pointer should point to the stack top symbol  $i$  since  $p$  is the state stored at (3).

We state two lemmas used to prove Theorem 13. Let  $\vdash_{(n)}$  denote the subrelation of  $\vdash$  derived from the rule  $(n)$ . The following lemma is immediate from the definition of  $\vdash_{(n)}$ .

► **Lemma 14.** For all  $q, q' \in Q_N$ ,  $w, w' \in \Sigma^*$ ,  $\gamma, \gamma' \in \Gamma^*$ ,

(a) 1. for each  $a \in \Sigma$ ,  $(q, aw, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(1)} (q', w, \#Z_0\gamma a\uparrow\mathbb{S})$  if  $q \xrightarrow{a}_N q'$ ,  
 2.  $\exists a \in \Sigma. q \xrightarrow{a}_N q' \wedge w = aw' \wedge \beta = Z_0\gamma a\uparrow$  if  $(q, w, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(1)} (q', w', \#\beta\mathbb{S})$ ,

(b) 1. for each  $b \in B_k$ ,  $(q, w, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(2)} (q', w, \#Z_0\gamma b\uparrow\mathbb{S})$  if  $q \xrightarrow{b}_N q'$ ,  
 2.  $\exists b \in B_k. q \xrightarrow{b}_N q' \wedge w = w' \wedge \beta = Z_0\gamma b\uparrow$  if  $(q, w, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(2)} (q', w', \#\beta\mathbb{S})$ .

In particular, letting  $\vdash_{(1),(2)} = \vdash_{(1)} \uplus \vdash_{(2)}$ , we obtain the following statement by repeating  $(a)_1$  and  $(b)_1$  zero or more times: For all  $v \in (\Sigma \uplus B_k)^*$ ,  $(q, g(v)w, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(1),(2)}^* (q', w, \#Z_0\gamma v\uparrow\mathbb{S})$  if  $q \xrightarrow{v}_N q'$ .

► **Lemma 15.** Suppose that  $q \xrightarrow{i}_N q'$ , and  $\gamma i$  is matching. Let  $m = \text{cnt}(\gamma i)$ . For all  $p \in Q_N$ ,  $w, w' \in \Sigma^*$  and  $\beta \in (\Gamma \uplus \{\mathbf{c}, \mathbb{S}, \uparrow\})^*$ , the following (a) and (b) are equivalent (see Appendix A for the proof):

(a)  $p = q'$ ,  $w = g((\gamma i)_{[m]})w'$ , and  $\beta = Z_0\gamma i\uparrow$ .

(b)  $(q, w, \#Z_0\gamma\uparrow\mathbb{S}) \vdash_{(3)} (W_{q'}, w, \#Z_0\gamma i\uparrow\mathbb{S}) \vdash \dots \vdash (p, w', \#\beta\mathbb{S})$ , where no ID with a state in  $Q_N$  appears in the calculation  $\dots$ .

**Proof of Theorem 13 (sketch).** For proving  $L(\alpha) \subseteq L(A_\alpha)$ , we take  $w \in L(\alpha)$  and  $v \in \mathcal{R}_k(\alpha)$  such that  $w = \mathcal{D}_k(v)$ . Decomposing  $v$  into  $v_0 n_1 v_1 \dots n_m v_m$  (where  $m = \text{cnt } v$ ), we obtain a transition sequence in the underlying NFA  $N$ , denoted by  $q_0 \xrightarrow{v_0}_N q_{(0)} \xrightarrow{n_1 v_1}_N q_{(1)} \xrightarrow{n_2 v_2}_N \dots \xrightarrow{n_m v_m}_N q_{(m)} \in F$ . We prove by induction on  $r = 0, \dots, m$  that  $A_\alpha$  can reach  $q_{(r)}$  while consuming  $z_r = g(v_0)g(v_{[1]})g(v_1) \dots g(v_{[r]})g(v_r)$  from the input and pushing  $y_r = v_0 n_1 v_1 \dots n_r v_r$  to the stack. Conversely, we suppose a calculation in  $A_\alpha$ , denoted by  $C_{(1)} = (q_0, w, \#Z_0\uparrow\mathbb{S}) \vdash \dots \vdash C_{(r)} \vdash \dots \vdash C_{(m)} = (p_m, \varepsilon, \#\beta_m\mathbb{S})$ , where  $p_m \in F$  and  $C_{(r)} = (p_r, w_r, \#\beta_r\mathbb{S})$  for each  $r \in \{1, \dots, m\}$ . By induction on  $r = 1, \dots, m$ , we extract an underlying transition  $q_0 \xrightarrow{\gamma_r}_N p_r$  step by step while maintaining the invariants  $\gamma_r \in (\Sigma \uplus B_k \uplus [k])^*$  and  $w = \mathcal{D}_k(\gamma_r)w_r$ , as long as  $p_r \in Q_N$  (the formal proof is available in the full version [13]). ◀

► **Corollary 16.** Every rew b describes an indexed language, but not vice versa.

**Proof.** The first half follows by Theorem 13 since 1N NSA and indexed grammars are equivalent [2]. The second half also follows since the class of CFLs is a subclass of indexed languages [1], and the class of rewbs and that of CFLs are incomparable [4]. ◀

In the case of a rew b  $\alpha$  without a captured reference (that is, one in which no reference  $\setminus i$  appears as a subexpression of an expression of the form  $(j \dots)_j$ ), we can transform  $A_\alpha$  into an NESAs  $A''_\alpha$  recognizing  $L(\alpha)$ , i.e., one that neither uses substacks nor pops its stack. First, we transform  $A_\alpha$  to an NSA without substacks (i.e., SA)  $A'_\alpha$ . Inspecting how substacks are used in  $A_\alpha$ , we can drop rules (12) and (16) in  $A'_\alpha$  because there is no captured reference in  $\alpha$ . We also remove the uses of substacks from rules (3) and (4), which correspond to calling, and rules (14), (15) and (17), which correspond to returning. Namely, while  $A_\alpha$ , upon a call, stores the substack  $eq'\mathbb{S}$  that consists of just the state  $q'$  where the control should return,  $A'_\alpha$  simply pushes  $q'$  to the stack top. That is, we remove (4), (15) and (17), and change (3) and (14) to the following (3') and (14'), respectively:

$$(3') \delta_N(q, i) \ni q' \implies \delta(q, c, Z\mathbb{S}) \ni (c_i, S, Zi q'\mathbb{S}), \quad (14') \delta(r_i, c, q\mathbb{S}) = \{(q, S, \mathbb{S})\}.$$

Furthermore, we transform  $A'_\alpha$  to an SA without stack popping (i.e., NESAs)  $A''_\alpha$ . Observe that  $A'_\alpha$  pops only when returning via (14') and popping a state that was pushed in a preceding call. Thus,  $A''_\alpha$ , rather than popping  $q'$ , leaves it on the stack, and has the modes  $c_i$ ,  $e_i$  and  $r_i$  skip all state symbols on the stack except the ones at the top. Here, we only need to modify  $e_i$  since  $A_\alpha$  already skips them at  $c_i$  and  $r_i$  (rules (6) and (13)). In short, we add the new rule (9\*) and change (14') to (14''), as follows:

$$(9^*) \delta(e_i, c, q) = \{(e_i, S, R)\}, \quad (14'') \delta(r_i, c, q\mathbb{S}) = \{(q, S, q\mathbb{S})\}.$$

This NESAs  $A''_\alpha$  whose transition function consists of the rules (1),(2),(3'),(5)–(9),(9\*), (10), (11), (13) and (14'') recognizes  $L(\alpha)$ . Therefore,

► **Corollary 17.** *Every rew without a captured reference describes a nonerasing stack language, but not vice versa.*<sup>6</sup>

Note that the converse of Corollary 17 fails to hold. In other words, there is a rew with a captured reference that describes a nonerasing stack language. The rew  $(1a)_1(2\backslash 1)_2\backslash 2$  is a simple counterexample. In addition, as shown later in Section 6, NESAs can recognize nontrivial language (hierarchy) with a captured reference such as Larsen's hierarchy [12].

## 5 A rew that describes a non-stack language

We just showed that every rew describes an indexed language and in particular every rew without a captured reference describes a nonerasing stack language. So, a natural question is whether every rew describes a (nonerasing) stack language. We show that the answer is *no*. That is, there is a rew that describes a non-stack language.

Ogden has proposed a pumping lemma for stack languages and shown that the language  $\{a^{n^3} \mid n \in \mathbb{N}\}$  is a non-stack language as an application (see [14], Theorem 2). A key point in the proof is that the exponential  $n^3$  of  $a$  is a cubic polynomial, and we can show that for every cubic polynomial  $f : \mathbb{N} \rightarrow \mathbb{N}$ , the language  $\{a^{f(n)} \mid n \in \mathbb{N}\}$  is also a non-stack language by the same proof. Thus, a rew that describes a language in this form is a counterexample. We borrow the technique in [6] (Example 1) which shows that the rew  $\alpha = ((1\backslash 2)_1(2\backslash 1a)_2)^*$  describes  $L(\alpha) = \{a^{n^2} \mid n \in \mathbb{N}\}$ . This follows since  $\mathcal{D}_k((\llbracket 12 \rrbracket_1 \llbracket 21a \rrbracket_2)^n) = a^{n^2}$  holds by recording the iteration count of the Kleene star,  $n$ , in the capture  $(2)_2$  as  $a^n$ , and extending the length by  $2n + 1$ , as shown below:

$$\begin{aligned} \mathcal{D}_k((\llbracket 12 \rrbracket_1 \llbracket 21a \rrbracket_2)^{n+1}) &= \mathcal{D}_k((\llbracket 12 \rrbracket_1 \llbracket 21a \rrbracket_2)^n \llbracket 12 \rrbracket_1 \llbracket 21a \rrbracket_2) = \mathcal{D}_k(\cdots \llbracket 2a^n \rrbracket_2 \llbracket 12 \rrbracket_1 \llbracket 21a \rrbracket_2) \\ &= \mathcal{D}_k(\cdots \llbracket 2a^n \rrbracket_2 \llbracket 1a^n \rrbracket_1 \llbracket 21a \rrbracket_2) = \mathcal{D}_k(\cdots \llbracket 2a^n \rrbracket_2 \llbracket 1a^n \rrbracket_1 \llbracket 2a^{n+1} \rrbracket_2) = \underline{\underline{a^{n^2}}} a^{2n+1} = a^{(n+1)^2}. \end{aligned}$$

The rew  $((1\backslash 4a)_1(2\backslash 3)_2(3\backslash 2a)_3(4\backslash 1\backslash 3)_4)^*$  describes  $\{a^{n(n+7)(2n+1)/6} \mid n \in \mathbb{N}\}$  and extends the length by a quadratic in  $n$  instead (see the full version [13] for the calculation). Thus,

► **Theorem 18.** *There exists a rew that describes a non-stack language.*

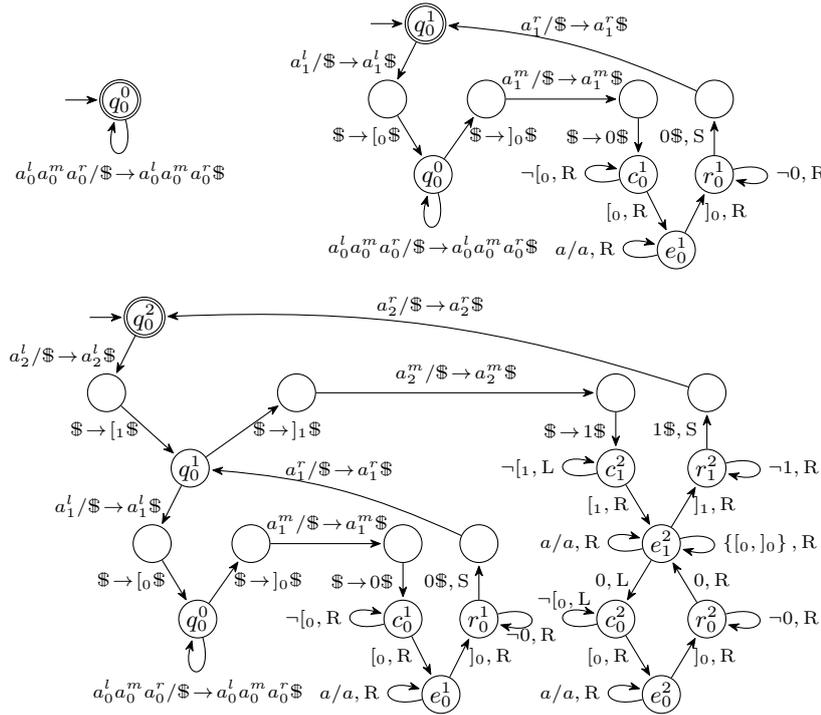
From this and Corollary 17, this rew needs a captured reference, in the sense that:

► **Corollary 19.** *There exists a rew that describes a language that no rew without a captured reference can describe.*

<sup>6</sup> For the latter part, we can take the language  $\{a^n b^n \mid n \in \mathbb{N}\}$  that can be described by an NESAs (see the full version [13]) but not by any rew [4].

**6 Larsen’s hierarchy is within the class of nonerasing stack language**

In this section, we construct an NESAs  $A_i$  that describes  $L(x_i)$ , where the rew  $x_i$  over the alphabet  $\Sigma = \{a_0^l, a_0^m, a_0^r, a_1^l, a_1^m, a_1^r, \dots\}$  is given by Larsen [12] and defined as follows:  $x_0 \triangleq (a_0^l a_0^m a_0^r)^*$ ,  $x_{i+1} \triangleq (a_{i+1}^l (x_i)_i a_{i+1}^m \setminus i a_{i+1}^r)^*$  ( $i \geq 0$ ). Our result implies that Larsen’s hierarchy is within the class of nonerasing stack languages. Since Larsen showed that no rew with its nested level less than  $i$  can describe  $L(x_i)$  [12], it also implies that for every  $i \in \mathbb{N}$ , there is a nonerasing stack language that needs a rew of nested level at least  $i$ .<sup>7</sup>



**Figure 1**  $A_0$  (upper left),  $A_1$  (upper right),  $A_2$  (lower).

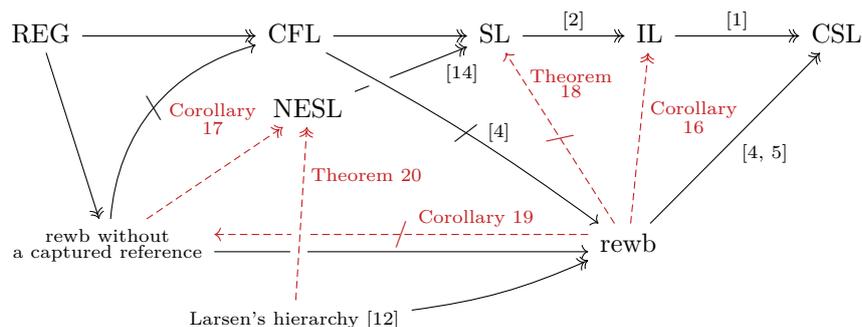
The NESAs  $A_i$  has the start state  $q_0^i$  which is also its only final state. Figure 1 depicts  $A_0$ ,  $A_1$ , and  $A_2$ .  $A_0$  is easy.  $A_1$  is obtained by connecting the eight states to  $q_0^0$  and making  $q_0^1$  the start/final state, as shown in the figure. The five states on the right handle the dereference of  $\setminus 0$  in  $x_1$ . That is, at  $c_0^1$ ,  $A_1$  first seeks the left-nearest  $[0$ , passes the control to  $e_0^1$ , checks the input string against the stack at  $e_0^1$ , passes the control to  $r_0^1$ , and at  $r_0^1$ , finally goes back to the right-nearest  $0$  which must be written on the stack top. In much the same way,  $A_2$  is obtained from  $A_1$  but we must be sensitive to the handling of the dereference of  $\setminus 1$  because  $A_2$  must handle the dereference of not only  $\setminus 1$  but also  $\setminus 0$  that appears in a string captured by  $[1]_1$  whereas no backreference appears in a string captured by  $[0]_0$  in the case of  $A_1$ . To deal with this issue, we connect the three new states  $c_0^2$ ,  $e_0^2$  and  $r_0^2$  to  $e_1^2$ . At  $e_1^2$ , if  $A_2$  encounters  $0$  in a checking,  $A_2$  suspends the checking and first goes to  $c_0^2$  to seek  $[0$ , goes to  $e_0^2$  to check the input against the stack by reading out a  $]0$  (no number appears

<sup>7</sup> Technically, Larsen [12] adopts a syntax that excludes unbound references, and so this implied result applies only to rews with no unbound references.

in this checking), and finally goes to  $r_0^2$  to go back to 0 which passed the control to  $c_0^2$ . We repeat this modification until  $A_i$  is obtained. (Thus,  $A_i$  has such states  $c_j^i, e_j^i, r_j^i$  for each  $j \in \{0, \dots, i-1\}$ .) Therefore,

► **Theorem 20.** *There exists an NESAs  $A_i$  that recognizes  $L(x_i)$ .*

## 7 Conclusions



■ **Figure 2** The inclusion relations between the classes.

In this paper, we have shown the following five results: (1) that every rew b describes an indexed language (Corollary 16), (2) in particular that every rew b without a captured reference describes a nonerasing stack language (Corollary 17), (3) however that there exists a rew b that describes a non-stack language (Theorem 18), (4) therefore that there exists a rew b that needs a captured reference (Corollary 19), and (5) finally that Larsen’s hierarchy  $\{L(x_i) \mid i \in \mathbb{N}\}$  given in [12] is within the class of nonerasing stack languages (Theorem 20). We have obtained the results by using three automata models, namely NESAs, SAs, and NSAs, and using the semantics of rewbs given in [15, 6] that treats a rew b as a regular expression allowing us to obtain the underlying NFA. Figure 2 depicts the inclusion relations between the classes mentioned in the paper. Here,  $A \rightarrow B$  stands for  $A \subseteq B$ ,  $A \twoheadrightarrow B$  for  $A \subsetneq B$ , and  $A \not\rightarrow B$  for  $A \not\subseteq B$ , respectively. A label on an arrow refers to the evidence. A red dashed arrow indicates a novel result proved in this paper, where for a strict inclusion, we show for the first time the inclusion itself in addition to the fact that it is strict.

As future work, we would like to investigate the use of the pumping lemma for rewbs without a captured reference that can be derived from the contraposition of our Corollary 17 and a pumping lemma for NESAs [14]. We expect it to be a useful tool for discerning which rewbs need captured references. Additionally, we suspect that our construction of NESAs in Theorem 20 is useful for not just  $x_i$  of [12] but also for more general rewbs that have only one  $\setminus i$  for each  $(i)_i$ , and we would like to investigate further uses of the construction.

## References

- 1 Alfred V Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM (JACM)*, 15(4):647–671, 1968.
- 2 Alfred V Aho. Nested stack automata. *Journal of the ACM (JACM)*, 16(3):383–406, 1969.
- 3 Alfred V. Aho. *Algorithms for finding patterns in strings*, pages 255–300. MIT Press, Cambridge, MA, USA, 1991.
- 4 Martin Berglund and Brink van der Merwe. Re-examining regular expressions with backreferences. *Theoretical Computer Science*, 940:66–80, 2023.

- 5 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14(06):1007–1018, 2003.
- 6 Dominik D Freydenberger and Markus L Schmid. Deterministic regular expressions with back-references. *Journal of Computer and System Sciences*, 105:1–39, 2019.
- 7 Seymour Ginsburg, Sheila A Greibach, and Michael A Harrison. One-way stack automata. *Journal of the ACM (JACM)*, 14(2):389–418, 1967.
- 8 Seymour Ginsburg, Sheila A Greibach, and Michael A Harrison. Stack automata and compiling. *Journal of the ACM (JACM)*, 14(1):172–201, 1967.
- 9 Takeshi Hayashi. On derivation trees of indexed grammars – An extension of the uvwxy-theorem. *Publications of the Research Institute for Mathematical Sciences*, 9(1):61–92, 1973.
- 10 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation, 2nd Edition*. Addison-Wesley, 2001.
- 11 John E. Hopcroft and Jeffrey D. Ullman. Nonerasing stack automata. *Journal of Computer and System Sciences*, 1(2):166–186, 1967.
- 12 Kim S Larsen. Regular expressions with nested levels of back referencing form a hierarchy. *Information Processing Letters*, 65(4):169–172, 1998.
- 13 Taisei Nogami and Tachio Terauchi. On the expressive power of regular expressions with backreferences, July, 2023. [arXiv:2307.08531](https://arxiv.org/abs/2307.08531).
- 14 William F Ogden. Intercalation theorems for stack languages. In *Proceedings of the first annual ACM symposium on Theory of computing*, pages 31–42, 1969.
- 15 Markus L Schmid. Characterising regex languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016.

## A Proof of Lemma 15

First, we recall the notation  $v_{[r]}$  (cf. Section 3) and explain a new notation  $v^{(r)}$  informally (see the full version [13] for the formal definitions of  $v_{[r]}$  and  $v^{(r)}$ ). Let  $k$  be a positive integer and  $v = v_0 n_1 v_1 \dots n_m v_m$  ( $m = \text{cnt } v$ ) a matching string over  $\Sigma \uplus B_k \uplus [k]$ . For each  $r = 1, 2, \dots$ , the notation  $v_{[r]}$  denotes the string which  $\mathcal{D}_k$  scans at the  $r^{\text{th}}$  number  $n_r$  and  $v^{(r)}$  the string immediately after the  $r^{\text{th}}$  replacement (also we let  $v^{(0)} = v$ ). For example, in the case of  $v = [1a [2b]_2 2]_1 1$ ,  $\mathcal{D}_k$  processes  $v$  as follows, therefore  $v_{[1]} = b$  and  $v_{[2]} = a[2b]_2 b$ :  $v^{(0)} = [1a [2b]_2 2]_1 1 \rightarrow v^{(1)} = [1a [2b]_2 b]_1 1 \rightarrow v^{(2)} = [1a [2b]_2 b]_1 abb \rightarrow abbabb$ . We can easily prove the following claim (see also the full version [13]): For each  $r \in \{0, 1, \dots, m\}$ ,

$$v^{(r)} = v_0 g(v_{[1]}) v_1 \cdots g(v_{[r]}) v_r n_{r+1} v_{r+1} \cdots n_m v_m. \quad (*)$$

We prepare some more notations for the proof. Let  $\Sigma_{\perp}^* \triangleq \Sigma^* \uplus \{\perp\}$  and for every  $w \in \Sigma_{\perp}^*$  and  $s \in \Sigma^*$ , let  $w/s$  denote the string  $w$  but with the suffix  $s$  erased if  $w$  ends with  $s$ , and otherwise  $\perp$ . To use this notation, we expand the set of all IDs  $I = Q \times \Sigma^* \times \{\#\}$  ( $\Gamma \uplus \{\mathfrak{c}, \$, \uparrow\}^* \{\mathfrak{S}\}$ ) to  $I_{\perp} \triangleq Q \times \Sigma_{\perp}^* \times \{\#\}$  ( $\Gamma \uplus \{\mathfrak{c}, \$, \uparrow\}^* \{\mathfrak{S}\}$ ), and we let  $C(w)$  denote an ID  $C = (\cdot, w, \cdots)$  and  $\vdash'_{(n)}$  denote the following binary relation over  $I_{\perp}$ :

$$\vdash'_{(n)} \triangleq \left\{ (C(w), C'(w/(a_i \cdots a_{i+\bar{d}-1}))) \mid C(a_i \cdots a_k) \vdash_{(n)} C'(a_{i+\bar{d}} \cdots a_k), w \in \Sigma_{\perp}^* \right\}.$$

In addition, we define  $\vdash' \triangleq \bigcup_n \vdash'_{(n)}$ . Then,  $\vdash \subseteq \vdash'$  is immediate and we show that the converse partially holds, in the sense that:

► **Lemma 21.** *For every string  $w \in \Sigma^*$  and  $w' \in \Sigma^*$ ,  $C(w) \vdash' C'(w')$  implies  $C(w) \vdash C'(w')$ .*

**Proof.** By the definition of  $\vdash'$ , there is  $(C(a_i \cdots a_k), C'(a_{i+\bar{d}} \cdots a_k)) \in \vdash$  such that  $w' = w/(a_i \cdots a_{i+\bar{d}-1})$ . By the definition of  $\vdash$ ,  $C(w) = C(a_i \cdots a_{i+\bar{d}-1} w') \vdash C'(w')$  holds. ◀

► **Definition 22.** Given  $C, C' \in I_{\perp}$ , we write  $C \models_{(n)} C'$  if  $C \vdash'_{(n)} C'$  and  $\forall j, C'' . C \vdash'_{(j)} C'' \implies j = n \wedge C'' = C'$ . We often omit the subscript  $(n)$  and simply write  $C \models C'$ . Note that  $C \models C'$  implies not only  $C \vdash' C'$  but also determinism:  $\forall C'' \in I_{\perp} . C \vdash' C'' \implies C' = C''$ .

► **Lemma 23.** Suppose that  $\gamma \in (\Sigma \uplus B_k \uplus [k])^*$ ,  $i \in [k]$ ,  $w \in \Sigma^*$ ,  $\beta \in (\Gamma \uplus \{\mathfrak{c}, \$\})^*$  and  $p \in Q_N \uplus \{e_i \mid i \in [k]\}$ . Let  $m = \text{cnt}(\gamma i) (\geq 1)$ . If  $\gamma i$  is matching,

$$(c_i, w, \#Z_0\gamma\text{cp}1\$\beta\$) \models \dots \models (r_i, w/g((\gamma i)_{[m]}), \#Z_0\gamma\text{cp}1\$\beta\$)$$

holds, where no ID with a state in  $Q_N$  appears in the calculation  $\dots$ .

**Proof.** In this proof, we sometimes write the stack representation  $\#\dots Z1\dots\$\$$  as  $\#\dots \uparrow Z\dots\$\$$  with the head-reversed arrow  $\uparrow$ . First, if  $\gamma \not\# [i]$ , it holds that

$$\begin{aligned} (c_i, w, \#Z_0\gamma\text{cp}1\$\beta\$) &\models^* (c_i, w, \#Z_0\uparrow\gamma\text{cp}\$\beta\$) \\ &\models (r_i, w, \#Z_0\uparrow\gamma\text{cp}\$\beta\$) \models^* (r_i, w, \#Z_0\gamma\text{cp}1\$\beta\$), \end{aligned}$$

and by  $(\gamma i)_{[m]} = \varepsilon$ , we have  $w = w/g((\gamma i)_{[m]})$ , as required. Henceforth, we assume that  $\gamma \ni [i]$  and the decomposition  $\gamma = \gamma_0[i\gamma_1$  ( $\gamma_1 \not\# [i]$ ). Moreover, we can further decompose  $\gamma_1 = \gamma_2[i\gamma_3$  ( $\gamma_2 \not\# [i]$ ,  $\gamma_3 \not\# [i]$ ) because  $\gamma i$  is matching. We prove by induction on  $m$ .

Case  $m = 1$ : By  $\text{cnt} \gamma = 0$ ,  $\gamma_2 \in (\Sigma \uplus B_k)^*$  follows. Letting  $w' \triangleq w/g(\gamma_2)$ , we have

$$\begin{aligned} (c_i, w, \#Z_0\gamma_0[i\gamma_1\text{cp}1\$\beta\$) &\models^* (c_i, w, \#Z_0\gamma_0[i\uparrow\gamma_1\text{cp}\$\beta\$) \models (e_i, w, \#Z_0\gamma_0[i\uparrow\gamma_1\text{cp}\$\beta\$) \\ &\models^* (e_i, w', \#Z_0\gamma_0[i\gamma_2[i\uparrow\gamma_3\text{cp}\$\beta\$) \models (r_i, w', \#Z_0\gamma_0[i\gamma_2[i\uparrow\gamma_3\text{cp}\$\beta\$) \\ &\models^* (r_i, w', \#Z_0\gamma_0[i\gamma_2[i\gamma_3\text{cp}1\$\beta\$). \end{aligned}$$

Therefore, the claim holds since no ID with a state in  $Q_N$  appears in this calculation and  $\gamma_2 = (\gamma i)_{[m]}$  follows from  $(\gamma i)^{(0)} = \gamma i = \gamma_0[i\gamma_2[i\gamma_3i$ ,  $\gamma_3 \not\# [i]$ .

Case  $\{1, \dots, m\} \implies m + 1$ : Let  $m_0 \triangleq \text{cnt} \gamma_0$  and  $l \triangleq \text{cnt} \gamma_2 (\geq 0)$ . Now,  $m_0 + l \leq m = \text{cnt} \gamma$  holds and we write  $\gamma_2 = \lambda_0 n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l$ . We also define  $\eta_r \triangleq \gamma_0[\lambda_0 n_{m_0+1} \dots \lambda_{r-1} n_{m_0+r}$  for each  $r \in \{1, \dots, l\}$ . By  $\eta_r$  being a prefix of  $\gamma i$  and Lemma 7,  $\eta_r$  is matching and  $(\eta_r)_{[m_0+r]} = (\gamma i)_{[m_0+r]}$ ,  $r \in \{1, \dots, l\}$  holds. In particular, it follows that  $n_{m_0+r} \neq i$  for every  $r$  (if there is  $r$  such that  $n_{m_0+r} = i$ ,  $\gamma_2 \supseteq \lambda_0 n_{m_0+1} \dots \lambda_{r-1} \ni [i]$  holds but this contradicts  $\gamma_2 \not\# [i]$ ). Thus, letting  $w_0 \triangleq w$ ,  $w'_r \triangleq w_{r-1}/g(\lambda_{r-1})$ ,  $w_r \triangleq w'_r/g((\eta_r)_{[m_0+r]})$  and  $w' = w_l/g(\lambda_l)$ , we have

$$\begin{aligned} &(c_i, w, \#Z_0\gamma\text{cp}1\$\beta\$) \\ &\models^* (c_i, w, \#Z_0\gamma_0[i\uparrow\lambda_0 n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models (e_i, w_0, \#Z_0\gamma_0[i\uparrow\lambda_0 n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models^* (e_i, w'_1, \#Z_0\gamma_0[i\lambda_0 n_{m_0+1} \uparrow \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models (c_{n_{m_0+1}}, w'_1, \#Z_0\gamma_0[i\lambda_0 \text{cr}_i \uparrow \$n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models^* (r_{n_{m_0+1}}, w_1, \#Z_0\gamma_0[i\lambda_0 \text{cr}_i \uparrow \$n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\quad \text{(by } \eta_1 \text{ being matching and induction hypothesis)} \\ &\models (E_{e_i, n_{m_0+1}}, w_1, \#Z_0\gamma_0[i\lambda_0 \mathfrak{c} \uparrow \$n_{m_0+1} \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models (L_{e_i, n_{m_0+1}}, w_1, \#Z_0\gamma_0[i\lambda_0 n_{m_0+1} \uparrow \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models (e_i, w_1, \#Z_0\gamma_0[i\lambda_0 n_{m_0+1} \uparrow \lambda_1 \dots n_{m_0+l} \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\models^* \dots \models^* (e_i, w_l, \#Z_0\gamma_0[i\lambda_0 n_{m_0+1} \lambda_1 \dots n_{m_0+l} \uparrow \lambda_l]_i \gamma_3 \text{cp} \$\beta\$) \\ &\quad \text{(by similar calculation and induction hypothesis)} \end{aligned}$$

$$\begin{aligned}
& \models^* (e_i, w', \#Z_0\gamma_0[i\lambda_0n_{m_0+1}\lambda_1\cdots n_{m_0+l}\lambda_l]_i \uparrow \gamma_3 \mathfrak{c} p \$i\beta \$) \\
& \models (r_i, w', \#Z_0\gamma_0[i\lambda_0n_{m_0+1}\lambda_1\cdots n_{m_0+l}\lambda_l]_i \uparrow \gamma_3 \mathfrak{c} p \$i\beta \$) \\
& \models^* (r_i, w', \#Z_0\gamma_0[i\lambda_0n_{m_0+1}\lambda_1\cdots n_{m_0+l}\lambda_l]_i \gamma_3 \mathfrak{c} p \uparrow \$i\beta \$),
\end{aligned}$$

and

$$\begin{aligned}
w' &= w/g(\lambda_0) g((\eta_1)_{[m_0+1]}) g(\lambda_1) \cdots g((\eta_l)_{[m_0+l]}) g(\lambda_l) \\
&= w/g(\lambda_0) g((\gamma^i)_{[m_0+1]}) g(\lambda_1) \cdots g((\gamma^i)_{[m_0+l]}) g(\lambda_l),
\end{aligned}$$

where no ID with a state in  $Q_N$  appears in this calculation. Here, we write

$$\gamma = \gamma_0[i\lambda_0n_{m_0+1}\lambda_1\cdots n_{m_0+l}\lambda_l]_i \gamma_3 = v_0 n_1 v_1 \cdots n_m v_m$$

and decompose its substrings as

$$v_{m_0} = \chi_0[i\lambda_0, \quad v_{m_0+l} = \lambda_l]_i \chi_1, \quad \text{and} \quad \gamma_3 = \chi_1 n_{m_0+l+1} v_{m_0+l+1} \cdots n_m v_m.$$

Then, by equation (\*), we can write  $(\gamma^i)^{(m)}$  as

$$v_0 \cdots \underbrace{\chi_0[i\lambda_0 g((\gamma^i)_{[m_0+1]}) v_{m_0+1} \cdots g((\gamma^i)_{[m_0+l]}) \lambda_l]_i}_{v_{m_0}} \underbrace{\chi_1 g((\gamma^i)_{[m_0+l+1]}) v_{m_0+l+1} \cdots g((\gamma^i)_{[m]}) v_m}_i.$$

That is, it holds that  $\gamma_3 \triangleq \chi_1 g((\gamma^i)_{[m_0+l+1]}) v_{m_0+l+1} \cdots g((\gamma^i)_{[m]}) v_m \not\equiv [i$  by  $\gamma_3 \not\equiv [i$ , and we obtain  $(\gamma^i)_{[m+1]} = \lambda_0 g((\gamma^i)_{[m_0+1]}) \lambda_1 \cdots g((\gamma^i)_{[m_0+l]}) \lambda_l$ , as shown above. Therefore, the claim holds for  $m+1$  since  $w' = w/g((\gamma^i)_{[m+1]})$ .  $\blacktriangleleft$

**Proof of Lemma 15.** For arbitrary  $w \in \Sigma^*$ , by Lemma 23,

$$\begin{aligned}
& (q, w, \#Z_0\gamma \uparrow \$) \vdash_{(3)} (W_{q'}, w, \#Z_0\gamma i \uparrow \$) \models_{(4)} (c_i, w, \#Z_0\gamma \mathfrak{c} q' \uparrow \$i \$) \\
& \models^* (r_i, w/g((\gamma^i)_{[m]}), \#Z_0\gamma \mathfrak{c} q' \uparrow \$i \$) \models_{(14)} (E_{q',i}, w/g((\gamma^i)_{[m]}), \#Z_0\gamma \mathfrak{c} \uparrow \$i \$) \\
& \models_{(15)} (L_{q',i}, w/g((\gamma^i)_{[m]}), \#Z_0\gamma i \uparrow \$) \models_{(17)} (q', w/g((\gamma^i)_{[m]}), \#Z_0\gamma i \uparrow \$) \quad (**)
\end{aligned}$$

holds. Assuming (a), we can replace  $\models$  in equation (\*\*) with  $\vdash$  by Lemma 21 because  $w/g((\gamma^i)_{[m]}) = w' \in \Sigma^*$  holds, and therefore, (b) follows. Supposing (b) conversely, we have  $(q, w, \#Z_0\gamma \uparrow \$) \vdash_{(3)} (W_{q'}, w, \#Z_0\gamma i \uparrow \$) \vdash' \cdots \vdash' (p, w', \#\beta \$)$ , where no ID with a state in  $Q_N$  appears in either this calculation or (\*\*) except in their leftmost and rightmost IDs. Therefore, their two calculations coincide by the determinism of  $\models$ . In particular, we obtain  $p = q'$ ,  $w' = w/g((\gamma^i)_{[m]})$  and  $\beta = Z_0\gamma i \uparrow$  by the equality of their rightmost IDs, and thus, (a) follows because  $w' \in \Sigma^*$ .  $\blacktriangleleft$



# OBDD(Join) Proofs Cannot Be Balanced

Sergei Ovcharov 

St. Petersburg State University, Russia

---

## Abstract

We study OBDD-based propositional proof systems introduced in 2004 by Atserias, Kolaitis, and Vardi that prove the unsatisfiability of a CNF formula by deduction of an identically false OBDD from OBDDs representing clauses of the initial formula. We consider a proof system  $\text{OBDD}(\wedge)$  that uses only the conjunction (join) rule and a proof system  $\text{OBDD}(\wedge, \text{reordering})$  (introduced in 2017 by Itsykson, Knop, Romashchenko, and Sokolov) that uses the conjunction (join) rule and the rule that allows changing the order of variables in OBDD.

We study whether these systems can be balanced i.e. every refutation of size  $S$  can be reassembled into a refutation of depth  $O(\log S)$  with at most a polynomial-size increase. We construct a family of unsatisfiable CNF formulas  $F_n$  such that  $F_n$  has a polynomial-size tree-like  $\text{OBDD}(\wedge)$  refutation of depth  $\text{poly}(n)$  and for arbitrary  $\text{OBDD}(\wedge, \text{reordering})$  refutation  $\Pi$  of  $F_n$  for every  $\alpha \in (0, 1)$  the following trade-off holds: either the size of  $\Pi$  is  $2^{\Omega(n^\alpha)}$  or the depth of  $\Pi$  is  $\Omega(n^{1-\alpha})$ . As a corollary of the trade-offs, we get that  $\text{OBDD}(\wedge)$  and  $\text{OBDD}(\wedge, \text{reordering})$  proofs cannot be balanced.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Proof complexity

**Keywords and phrases** Proof complexity, OBDD, lower bounds, depth of proofs

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.72

**Acknowledgements** The author is grateful to his supervisor Dmitry Itsykson for a suggesting of the problem, for the many fruitful discussions of it and for his active help in the preparation of this paper.

## 1 Introduction

The paper devotes to propositional proof complexity theory. Propositional proof systems are used for certifying that a given CNF formula is unsatisfiable. Investigation of propositional proof systems is highly connected with the construction of solvers for the Boolean satisfiability problem (SAT-solvers). The execution protocol of a SAT solver running on an unsatisfiable formula may be considered as a certificate of unsatisfiability. Every SAT solver is based on some proof system. For example, CDCL solvers are based on Resolution [3], Pseudo Boolean solvers are based on Cutting Planes [8], OBDD-solvers are based on OBDD-based proof systems [2].

The minimal refutation size of a formula is a natural lower bound on the running time of the corresponding SAT-Solvers. In this paper, we also study the depth of refutations, i.e. the length of the longest path from a clause of a refuted formula to a contradiction. The depth is a very natural but not a much-studied measure of the proofs. The minimal depth of a refutation is a lower bound on the parallel running time of the corresponding solver.

### Balancing proof systems

We consider only *refutational* proof systems. Each refutational proof system  $\Pi$  operates with *proof lines*, and each proof line is a Boolean predicate represented in some fixed way. Initially, all clauses of refuted formulas are represented by proof lines and new proof lines may be derived using a finite set of inference rules. The goal is to derive an identically false proof line. Every refutational proof system is defined by the type of predicates that may be



© Sergei Ovcharov;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 72; pp. 72:1–72:13

Leibniz International Proceedings in Informatics



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

used as proof lines and by the list of inference rules. For example, in Resolution proof lines are clauses, in Cutting Planes [10] proof lines are linear inequalities with integer coefficients and Boolean variables, in Frege proof systems proof lines are propositional formulas, etc.

The *size* of a refutation is the total size of representations of all used proof lines.

Every refutation can be represented as a directed acyclic graph with one source corresponding to a contradiction and sinks corresponding to clauses of a refuted formula, and every proof line is obtained by the descendants using the inference rule. The *depth* of the refutation is the depth of the corresponding graph i.e. the length of the longest path from the source to a sink.

We say that proofs in some proof system can be balanced if it is always possible to reassemble each refutation in such a way that its depth becomes logarithmic in its size (perhaps with a polynomial-size increase).

The question of whether Resolution proofs can be balanced is trivial. Indeed, consider the formula  $(x_1 \vee \dots \vee x_n) \wedge (\neg x_1) \wedge \dots \wedge (\neg x_n)$ . It is easy to see that every refutation of this formula must have the depth at least  $n$ , therefore, Resolution refutations cannot be balanced in the general case. Urquhart [16] studied if refutations of  $O(1)$ -CNF formulas can be balanced for which the question is less trivial. It was proven that there exists a family of 3-CNF formulas  $F_{n_i}$  with  $n_i$  variables having a Resolution refutation of polynomial size but every its refutation must have depth  $\Omega(n_i / \log n_i)$ . Therefore Resolution refutations cannot be balanced even for  $O(1)$ -CNF formulas.

Atserias, Bonet, and Levy [1] proved that Cutting Planes proofs cannot be balanced either. However, it is known that refutations in Frege systems can be balanced (see, for instance, [15]).

### OBDD-based proof systems

In 1986 Bryant [6] proposed an important way to represent a Boolean function. Every such function can be represented as a branching program with two sinks so that variables on every path from the source to a sink appear in the same order  $\pi$ . Such representation is called *Ordered Binary Decision Diagram* (OBDD or  $\pi$ -OBDD if we need to specify that the variables are ordered according to  $\pi$ ). The restriction on a variables order allows us to perform many useful operations with OBDD efficiently e.g. check satisfiability, compute the conjunction of two OBDDs (given they use the same variable order), etc [13].

Atserias, Kolaitis, and Vardi [2] introduced an OBDD-based refutational proof system. Among them, we are most interested in OBDD( $\wedge$ ). OBDD( $\wedge$ ) represents clauses of an unsatisfiable formula as  $\pi$ -OBDDs for some order  $\pi$  and the only refutation rule allows deriving the conjunction of two OBDDs which were derived earlier. The size of the refutation is the total size of the OBDDs in it.

Itsykson, Knop, Romashchenko, Sokolov [12] proposed the OBDD( $\wedge$ ,reordering) proof system. OBDD( $\wedge$ ,reordering) is obtained from OBDD( $\wedge$ ) by adding the reordering derivation rule that allows changing variables order of the derived OBDDs. While now OBDDs in the refutation may use different variable orders, the conjunction rule can be only applied to OBDDs that use the same variable order (otherwise it would be NP-hard to verify the correctness of such rule, see [14], Lemma 8.14).

Notice that the formula  $(x_1 \vee \dots \vee x_n) \wedge (\neg x_1) \wedge \dots \wedge (\neg x_n)$  that we considered above has a tree-like OBDD( $\wedge$ ) refutation of polynomial size and logarithmic depth.

For both the Resolution and the Cutting Planes proof systems there exist a family of formulas for which a refutation of small depth does not exist at all. We emphasize that it is not the case for OBDD-based proof systems. Indeed, every CNF formula with  $m$  clauses has

a tree-like OBDD( $\wedge$ ) refutation of the depth  $\log(m)$ ; the graph of this refutation is a full binary tree with  $m$  leaves. Note that the size of this proof can differ dramatically from the size of the minimum refutation. Hence in the notion of balancing we require that the size of the balanced proof should be bounded by a polynomial from the size of the initial proof.

### Our contribution

In Theorem 12 we construct a family of unsatisfiable formulas  $F_n$  having  $\text{poly}(n)$  size tree-like OBDD( $\wedge$ ) refutations such that the following size vs depth trade-offs holds. For every  $\alpha \in (0, 1)$ , any OBDD( $\wedge$ , reordering) refutation of  $F_n$  of depth  $O(n^{1-\alpha})$  requires size at least  $2^{\Omega(n^\alpha)}$ . Hence we prove that dag-like and tree-like OBDD( $\wedge$ ), OBDD( $\wedge$ , reordering) proofs cannot be balanced.

Formulas for which the trade-offs hold are the Pebbling formulas based on the grid graphs  $\text{Peb}(\text{Grid}_n)$ . Pebbling formulas are a well-studied family of formulas ([5], [16], [4]). Moreover, they were used for proving Resolution depth lower bounds in [16]. However, usually, they are used together with Pebbling games and Pebbling numbers of graphs. This is not the case for our result since we rely significantly on the structure of the grid graphs (including self-similarity and expansion) by themselves and do not use Pebbling games.

In Section 2 we define the main notions. In Subsection 3.1 we prove the OBDD size lower bounds for some set of *hard*  $\text{Peb}(\text{Grid}_n)$  subformulas. In Subsection 3.2 we prove the mentioned size vs depth trade-offs.

### Open question

It would be interesting to study the similar questions for OBDD( $\wedge$ , weakening) proof system which is obtained from OBDD( $\wedge$ ) by adding the *weakening* rule. The weakening rule allows deriving from an OBDD any its semantical implication represented by OBDD in the same order.

## 2 Preliminaries

► **Definition 1** (Branching Program). *Let  $X = \{x_1, \dots, x_n\}$  be a set of Boolean variables. A branching program is a directed acyclic graph with one node with indegree 0 (source) several inner nodes with outdegree 2 and two nodes with outdegree 0 (sinks). Every node except sinks is labeled with some variable from  $X$ , one of its outgoing edges is labeled with 0 and the other one is labeled with 1. One sink is labeled with 0 and the other one is labeled with 1.*

Every branching program represents some Boolean function of  $n$  variables. To compute a value of the function on input  $x_1 = a_1, \dots, x_i = a_i, \dots, x_n = a_n$  we start a path from the source, and for every vertex labeled with variable  $x_i$  we continue the path along the edge labeled with  $a_i$ , such a path reaches a sink and the label of this sink is the value of the function.

► **Definition 2** (Ordered Binary Decision Diagram(OBDD)). *A branching program is called OBDD if variables on every path from the source to sinks appear according to some fixed order of variables.*

*Sometimes we write  $\pi$ -OBDD instead of OBDD to emphasize that variables appear according to the order of variables  $\pi$ .*

## 72:4 OBDD(Join) Proofs Cannot Be Balanced

The order restriction in OBDDs allows to perform many useful operations on OBDDs efficiently e.g. minimize, check satisfiability, compute the conjunction of two OBDDs given they have a same order of variables, etc. [13].

Let us define a propositional proof system  $\text{OBDD}(\wedge, \text{reordering})$ .

► **Definition 3** ( $\text{OBDD}(\wedge, \text{reordering})$ ). *Let  $\varphi = \bigwedge_i C_i$  be an unsatisfiable CNF formula. A refutation of  $\varphi$  is a sequence of OBDDs  $D_1, D_2, \dots, D_t$  such that  $D_t$  is the constant false OBDD and for all  $1 \leq i \leq t$  the diagram  $D_i$  either represents a clause of  $\varphi$  or obtained from the previous  $D_j$ 's by one of the following derivation rules.*

- **Conjunction (or join)** rule allows deriving an  $\pi$ -OBDD for  $D_1 \wedge D_2$  from  $\pi$ -OBDDs  $D_1$  and  $D_2$ . We emphasize here that the conjunction rule can be only applied to OBDDs with the same order of variables.
- **Reordering** rule allows deriving an OBDD  $B$  from an equivalent OBDD  $A$  (note that  $A$  and  $B$  may use different variable orders).

*The size of a refutation is the sum of the sizes of the OBDDs from it.*

*Every  $\text{OBDD}(\wedge, \text{reordering})$  refutation can be represented as a directed acyclic graph (DAG) in which nodes are labeled with OBDDs from the refutation such that each sink is labeled with a OBDD for some clause of  $\varphi$ , the source is labeled with the constant false OBDD, and an OBDD in every inner node is the result of the application of some derivation rule to the OBDDs from the descendants.*

*A refutation is called tree-like if every node except the source has indegree one.*

*The depth of a refutation is the length of the longest path from the source to a sink.*

*We call a refutation a  $\pi$ -OBDD( $\wedge$ ) refutation if all OBDDs have the same order (i.e. no reordering rule was applied).*

Note, that in order to call  $\text{OBDD}(\wedge, \text{reordering})$  a proof system (in the sense of Cook-Reckhow [9]) we need to be able to efficiently check if some OBDD is the result of an application of the derivation rules to some others OBDDs. Fortunately, the restriction on a variable order allows us to do that, as we mentioned before (for the details see [12]).

► **Definition 4** (Pebbling formulas (see for instance [16])). *Let  $G = (E, V)$  be a directed acyclic graph. We associate with each node of  $G$  a distinct Boolean variable  $x$ ; we will identify nodes and the associated variables. The  $\text{Peb}(G)$  formula is the conjunction of the following clauses:*

- $(\neg u_1 \vee \dots \vee \neg u_n \vee v)$ , where  $v \in V$  and  $\{u_1, \dots, u_n\}$  is the set of all nodes such that edge  $(u_i, v) \in E$ . We denote this clause by  $(u_1, \dots, u_n \rightarrow v)$ . Note that if  $v$  is a source of the graph then  $n = 0$ . We call such clauses first type clauses.
- $(\neg v)$ , where  $v$  is a sink. We call such clauses second type clauses.

Note that for every directed acyclic graph  $G$  the formula  $\text{Peb}(G)$  is unsatisfiable.

Main goal of our work is to prove that  $\text{OBDD}(\wedge)$  and  $\text{OBDD}(\wedge, \text{reordering})$  refutations cannot be balanced. In order to do it we construct a family of CNF formulas such that the formulas have small  $\text{OBDD}(\wedge)$  refutations but they do not have refutations with small size and depth simultaneously.

► **Lemma 5** ([7]). *For every directed acyclic graph  $G$  and for every order of variables  $\pi$  formula  $\text{Peb}(G)$  has tree-like  $\pi$ -OBDD( $\wedge$ ) refutation of size  $O(|V|^2)$  and depth  $O(|V|)$ .*

**Proof.** See Appendix A. ◀

► **Definition 6** (Graph  $\text{Grid}_n$ ). Let  $\text{Grid}_n$  be a graph of the  $(n-1) \times (n-1)$  grid, with edges directed top to bottom and left to right.

In other words, the set of vertices is

$$V_n = \{(i, j), i, j \in [n]\}$$

and the set of edges is

$$E_n = \{((i, j), (i+1, j)) \mid i \in [n-1], j \in [n]\} \cup \{((i, j), (i, j+1)) \mid i \in [n], j \in [n-1]\}$$

► **Corollary 7.** Formula  $\text{Peb}(\text{Grid}_n)$  has  $\pi$ -OBDD( $\wedge$ ) refutation of the size  $O(n^4)$  and of the depth  $O(n^2)$  for every variable ordering  $\pi$ .

**Proof.** Follows from Lemma 5. ◀

Now, in order to prove that OBDD(join) proofs cannot be balanced, it is sufficient to prove size vs. depth trade-offs for refutations of  $\text{Peb}(\text{Grid}_n)$ . We prove such trade-offs in Theorem 12 but we still need several auxiliary lemmas.

► **Lemma 8** (Folklore). Let  $G$  be a directed acyclic graph with only one sink. Then  $\text{Peb}(G)$  minimal unsatisfiable i.e. a conjunction of every proper subset of its set of clauses is satisfiable.

**Proof.** See Appendix B. ◀

► **Notation 9.** For a graph  $G(V, E)$  (directed or undirected) and for two disjoint sets  $A, B \subset V$  denote by  $E(A, B)$  the set of edges with one end in  $A$  and the other one in  $B$ .

Note that for directed graphs we include in  $E(A, B)$  both the edges directed from  $A$  to  $B$  and the edges directed from  $B$  to  $A$ .

► **Definition 10** (Graph expansion [5]). Expansion of the graph  $G(V, E)$  is the minimum value of  $|E(U, V \setminus U)|$  among all subsets  $U \subset V$  such that  $\frac{|V|}{3} \leq |U| \leq \frac{2|V|}{3}$ .

► **Lemma 11** (Folklore).  $e(\text{Grid}_n) \geq \frac{1}{4}n$ .

**Proof.** Consider an arbitrary subset  $U \subset V$  such that  $\frac{1}{3}|V| \leq |U| \leq \frac{2}{3}|V|$ .

Assume that there are at least  $\frac{n}{4}$  columns of the grid containing nodes from both  $U$  and  $V \setminus U$ . Then there is at least one pair of incident nodes in every such column with one node in  $U$  and the other one in  $V \setminus U$ . Then the edges between the vertices from such pairs lie in  $E(U, V \setminus U)$ . Thus,  $|E(U, V \setminus U)| \geq \frac{n}{4}$ .

Now assume that there are at least  $\frac{3n}{4}$  columns lying completely in  $U$  or in  $V \setminus U$ . Since  $\frac{|V|}{3} \leq |U| \leq \frac{2|V|}{3}$ , there is at least one column completely lying in  $U$  and there is at least one column completely lying in  $V \setminus U$ . Therefore, in each row, there is at least one pair of incident nodes with one node in  $U$  and the other one in  $V \setminus U$ . In this case  $|E(U, V \setminus U)| \geq n$ . ◀

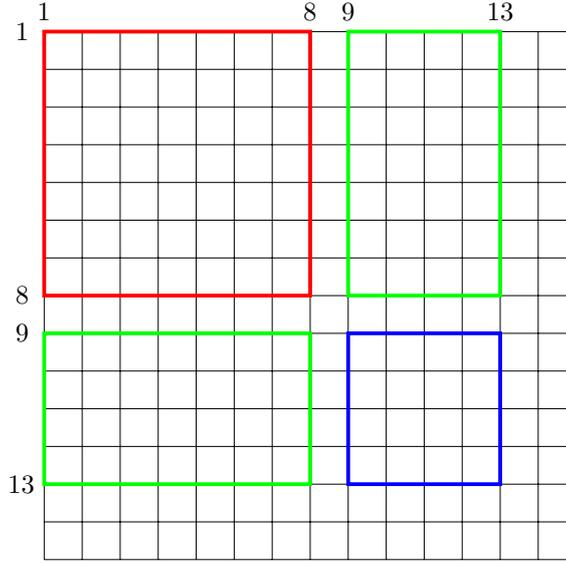
We want to point out that  $\text{Grid}_n$  graphs are not expanders in the conventional sense (see for example [11]) since  $\text{Grid}_n$  graph has  $n^2$  nodes but  $e(\text{Grid}_n) = \Theta(n)$  (Lemma 11 shows only that  $e(\text{Grid}_n) = \Omega(n)$  but upper bounds for  $e(\text{Grid}_n)$  are trivial).

### 3 Depth vs size trade-offs

In this section we prove our main result.

► **Theorem 12.** For every  $\alpha \in (0, 1)$  and for every OBDD( $\wedge$ , reordering) refutation of  $\text{Peb}(\text{Grid}_n)$  at least one of the following holds:

- the depth of the refutation is  $\Omega(n^\alpha)$ ;
- the size of the refutation is  $2^{\Omega(n^{1-\alpha})}$ .



■ **Figure 1** Possible partition for  $n = 15$ ,  $m = 13$ ,  $k = 5$ .

By a *configuration* we mean a conjunction of a subset of the set of  $\text{Peb}(\text{Grid}_n)$  clauses. In Subsection 3.1 we will show that some *hard* configurations cannot be represented by a small OBDD (see Lemma 13). Loosely speaking we are interested in configurations in which at least one clause from the top left part of the grid is missing and that contain many clauses from the bottom right part.

In Subsection 3.2 we finish the proof of Theorem 12 using Lemma 13. Namely, we show that either it is possible to find a hard configuration in the proof graph or the depth of the proof is large.

### 3.1 Configurations that are hard for OBDDs

Consider an arbitrary order  $\pi$  of variables of the formula  $\text{Peb}(\text{Grid}_n)$ .

Let  $\{x_0, y_0, x_1, y_1\} \subset [n]$ . We denote by  $[x_0, x_1] \times [y_0, y_1]$  the induced subgraph on the vertices set  $\{(x, y) \mid x_0 \leq x \leq x_1, y_0 \leq y \leq y_1\}$ .

Let  $m \in [n - 1]$ . We divide the nodes of the subgraph  $[1, m] \times [1, m]$  into the four parts:

- $[1, m - k] \times [1, m - k]$ ,
- $[m - k + 1, m] \times [m - k + 1, m]$ ,
- $[1, m - k] \times [m - k + 1, m]$ ,
- $[m - k + 1, m] \times [1, m - k]$ ,

where  $k \in [m]$  (see Fig. 1).

We divide variables associated with the vertices of  $[m - k + 1, m] \times [m - k + 1, m]$  into two equal (or differing by at most 1) parts in such a way that each variable from the first part appears in the order  $\pi$  before each variable of the second part. We denote the first part by  $A_\pi$  and the second by  $B_\pi$ . Recall that we identify each variable with the associated node and assume that  $A_\pi, B_\pi \subset V(\text{Grid}_n)$ .

Subgraph  $[m - k + 1, m] \times [m - k + 1, m]$  is isomorphic to the  $\text{Grid}_k$  and  $(|A_\pi| - |B_\pi|) \leq 1$ ,  $|A_\pi| + |B_\pi| = k^2$ , hence  $|E(A_\pi, B_\pi)| \geq \frac{k}{4}$  by Lemma 11.

Since the graph is directed, each edge in  $E(A_\pi, B_\pi)$  is directed from  $A_\pi$  to  $B_\pi$  or from  $B_\pi$  to  $A_\pi$ . Let us consider the direction with the majority of the edges. We denote the set of the corresponding edges by  $E_{0,\pi}$ ,  $|E_{0,\pi}| \geq |E(A_\pi, B_\pi)|/2 \geq \frac{k}{8}$ . All edges in  $E_{0,\pi}$  are directed from  $A_\pi$  to  $B_\pi$  or  $B_\pi$  to  $A_\pi$ . This gives us two cases that we consider later.

Using the following procedure we remove some of the edges from  $E_{0,\pi}$  to form a matching (that we will denote by  $E_{1,\pi}$ ).

While  $E_{0,\pi}$  is not empty:

- Choose an arbitrary edge  $e \in E_{0,\pi}$ , add it to  $E_{1,\pi}$ , remove it from  $E_{0,\pi}$ .
- If  $E_{0,\pi}$  still contains edges adjacent to  $e$  then we remove them. Since every node in  $\text{Grid}_n$  has degree at most 4 then we remove at most 6 edges per step.

We obtain matching  $E_{1,\pi} \subset E_{0,\pi}$  and  $|E_{1,\pi}| \geq \frac{|E_{0,\pi}|}{7} \geq \frac{k}{56}$ .

We call a node *special* if it is an head of some edge from  $E_{1,\pi}$ . There are  $|E_{1,\pi}|$  special nodes since  $E_{1,\pi}$  is a matching.

Using the following procedure we choose a subset  $W_\pi$  of the set of all special nodes such that the distance between any two nodes from  $W_\pi$  is at least 7.

While there are special nodes:

- Choose an arbitrary special node  $v$  that is not removed at the previous steps and add it to  $W_\pi$ .
- Remove all special nodes at the distance at most 6 from  $v$ . We removed at most  $(6 + 1 + 6)^2 = 169$  nodes, since all removed nodes are in the square of the size  $13 \times 13$  and with the center in  $v$ .

Since at each step we remove at most 169 special nodes and add one to  $W_\pi$ ,  $|W_\pi| \geq |E_{1,\pi}|/169 \geq k/9464$ .

For a node  $v = (i_0, j_0) \in V(\text{Grid}_n)$  we denote by  $B(v)$  the set  $\{(i, j) \in V(\text{Grid}_n) \mid |i - i_0| \leq 1, |j - j_0| \leq 1\}$  (i.e. the ball in the  $l_\infty$  metric). We refer to  $B(v)$  as a *ball* although it is not a ball in the graph distance sense.

For every node  $w \in W_\pi$  there is a unique edge in  $E_{1,\pi}$  with an endpoint  $w$ . Let us denote the set of start-points of such edges by  $U_\pi$ .

► **Lemma 13.** *Let  $\varphi$  be a conjunction of some subset of  $\text{Peb}(\text{Grid}_n)$  clauses. Suppose at least one clause of first type associated with a variable from the  $[1, m - k - 1] \times [1, m - k - 1]$  is not included in  $\varphi$ . Also, suppose that  $\varphi$  contains exactly  $d$  clauses associated with variables from  $W_\pi$ . Then the minimum size of an  $\pi$ -OBDD for  $\varphi$  is at least  $2^d$ .*

**Proof.** To prove that the size of any  $\pi$ -OBDD representation of  $\varphi$  is at least  $2^d$  it is sufficient to split  $\pi$  into two consequent parts and define  $2^d$  substitutions into the variables of the first part such that applications of them to  $\varphi$  lead to  $2^d$  different Boolean functions. To show that two substitutions  $\rho_0$  and  $\rho_1$  lead to two different functions we just define a substitution  $\rho$  into the second part of variables such that  $\varphi|_{\rho_0 \circ \rho} \neq \varphi|_{\rho_1 \circ \rho}$ .

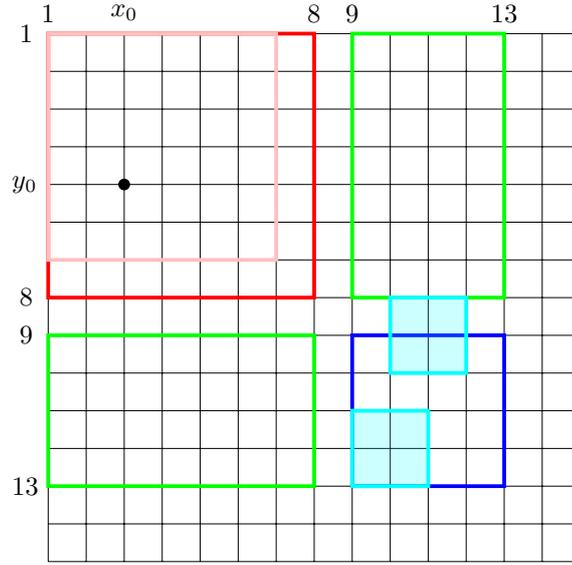
We have already divided the variables of  $[m - k + 1, m] \times [m - k + 1, m]$  into two parts  $A_\pi$  and  $B_\pi$  according to  $\pi$ . Let us fix a partition of  $\pi$  into two parts such that  $A_\pi$  lies in the first and  $B_\pi$  in the second.

Let  $W'_\pi$  be the nodes from  $W_\pi$  whose associated clauses are in  $\varphi$ . Let  $U'_\pi$  be the nodes from  $U_\pi$  connected with  $W'_\pi$ . Let  $W'_\pi = \{w_1, \dots, w_d\}$  and  $U'_\pi = \{u_1, \dots, u_d\}$ .

Substitutions we are defining differ only on a small set of nodes. To every variable outside this set each substitution assigns a fixed value (values may differ for different variables); we now define these values. Let  $z = (x_0, y_0)$  be an arbitrary node from  $[1, m - k - 1] \times [1, m - k - 1]$  whose first type clause is missing from  $\varphi$  (see Fig. 2).

- **Assignment to the variables in  $[1, n] \times [1, n] \setminus [x_0, n] \times [y_0, n]$ :**

To the variables whose corresponding nodes are strictly to the left or strictly at the top from  $z$  all substitutions assign 1. Note that all first-type clauses corresponding to the nodes in which we substitute 1 are satisfied (follows trivially from the definition of the  $\text{Grid}_n$  clauses).



■ **Figure 2** Balls  $B(w_i)$  are shown with light blue; their centers lie in  $[m - k + 1, m] \times [m - k + 1, m]$  (blue rectangle). Black bold point is  $(x_0, y_0)$ , it lies in  $[1, m - k - 1] \times [1, m - k - 1]$  (pink rectangle).

■ **Assignment to the variables in  $[x_0, n] \times [y_0, n] \setminus \bigcup_i B(w_i)$ :**

Consider the rectangle  $[x_0, n] \times [y_0, n]$ . It contains square  $[m - k, n] \times [m - k, n]$  and, therefore, all balls  $B(w_i)$  for  $i \in [d]$  (their centers lie in  $[m - k + 1, m] \times [m - k + 1, m]$ ). Let  $L = [x_0, n] \times [y_0, n] \setminus \bigcup_i B(w_i)$ .

▷ **Claim 14.** For every node  $v \in L$  there exists a directed path from  $z$  to  $v$  that lies completely in  $L$ .

*Proof.* Let us first note that for every  $i \in [d]$  and for every  $(x, y) \in B(w_i)$  it holds that  $x > x_0, y > y_0$ . Indeed,  $w_i \in [m - k + 1, m] \times [m - k + 1, m]$  hence  $(x, y) \in [m - k, m] \times [m - k, m]$  but  $z = (x_0, y_0) \in [1, m - k - 1] \times [1, m - k - 1]$ .

It is sufficient to prove that for every  $x \in L \setminus \{z\}$  at least one of its immediate predecessors lies in  $L$ . If we prove it we can build a desired path by induction. Suppose that there is a node  $x \in L \setminus \{z\}$  such that its immediate predecessors lie outside of  $L$ . Hence each of them lies either in  $\bigcup_i B(w_i)$  or in  $V(\text{Grid}_n) \setminus [x_0, n] \times [y_0, n]$ .

Firstly, consider the case in which one of the predecessors lies outside the  $[x_0, n] \times [y_0, n]$ . Then coordinates of  $x$  look like  $(x_0, *)$  or  $(*, y_0)$ . Without loss of generality assume that  $x = (x_0, h)$  for some  $h \geq y_0$ . Then there is the following path:  $z = (x_0, y_0), (x_0, y_0 + 1), \dots, (x_0, h) = x$  between  $z$  and  $x$ . All its nodes lie in  $[x_0, n] \times [y_0, n]$  and none of them lie in  $\bigcup_i B(w_i)$  due to the restriction on the coordinates of vertices from the balls that we mentioned earlier. Hence, this case is impossible.

Therefore its predecessors lie in  $\bigcup_i B(w_i)$ . But one of its predecessors is above  $x$  and the other is to the left of  $x$ . Hence they lie in different balls (otherwise  $x$  would also lie in the ball, and this contradicts the assumption  $x \in L$ ). Hence the distance between the balls is at most 2 which contradicts the construction from the beginning of the subsection. This concludes the proof of the claim. ◁

All the substitutions assign 0 to the variables in  $L$ . Now we check that no first-type clause from  $\varphi$  is falsified after that. There is no  $z$ 's clause in  $\varphi$ . For every other node  $z' \in L$  there is a path from  $z$  to  $z'$  in  $L$ . We assign 0 to the nodes on the path hence we assign 0 to some immediate predecessor of  $z'$  hence its clause is satisfied (again, by the definition of the clause).

- We have already defined the substitutions into all variables except  $\bigcup_i B(w_i)$ . Fix  $j \in [d]$ . There are two nodes with edges going from them to  $w_j$ . One of them is  $u_j$ . We denote the other one by  $r_j$ . Note that  $\{u_j, r_j\} \subset B(w_j)$ . Every substitution will assign 1 to  $r_j$ . Hence its first-type clause is always satisfied. To the nodes  $B(w_j) \setminus \{u_j, w_j, r_j\}$  we always substitute 0. We need to check that it will not falsify their clauses. It is easy to see that each node from  $B(w_j) \setminus \{u_j, w_j, r_j\}$  has at least one immediate predecessor in  $L \setminus \{w_i, u_i, r_i \mid i \in [d]\}$  (see Fig. 3). But we assign zeros to the variables from this set. Hence clauses corresponding to the  $B(w_j) \setminus \{u_j, w_j, r_j\}$  are always satisfied (see Fig. 3).

Note that we substitute 0 to the sink. Indeed, if the sink does not lie in any ball, then it lies in  $L$ . Hence we substitute 0 to it. Otherwise, the sink lies in some ball; denote the center of this ball by  $(x_1, y_1)$ . Then  $x_1 \leq n - 1$  and  $y_1 \leq n - 1$ . But then the sink is the most right bottom variable of the ball, hence it is substituted with 0.

Therefore, the second-type clause corresponding to the sink is satisfied.

At this point, we have defined substitutions on the set where their values coincide. Now we define substitutions to the remaining nodes i.e.  $\{w_i, u_i \mid i \in [d]\}$ . We need to consider to cases: whether  $\{u_j \mid j \in [d]\}$  or  $\{w_j \mid j \in [d]\}$  lie in the first part of the variable order.

**Case 1:** Suppose that  $\{u_j \mid j \in [d]\}$  lies in the first part of the variable order. Consider all possible substitutions of zeros and ones into the  $\{u_j \mid j \in [d]\}$ . There are  $2^d$  such substitutions. Note that every node from  $\{u_j \mid j \in [d]\}$  has an immediate predecessor from  $L \setminus \{u_j, w_j, r_j \mid j \in [d]\}$  hence its clause is satisfied.

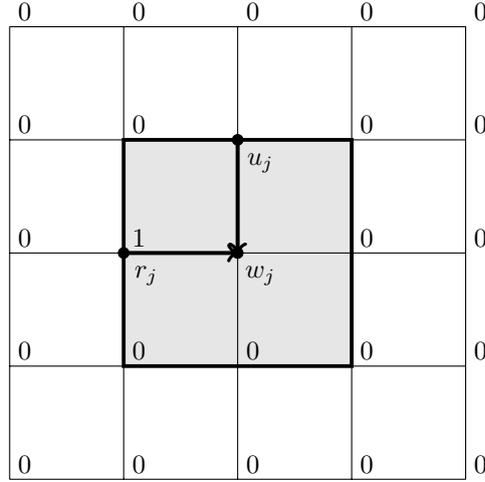
We show that we can separate any two such substitutions  $\rho_0 \neq \rho_1$  by some substitution  $\rho$  with support  $W'$ . There exists  $j_0 \in [d]$  such that  $\rho_0(u_{j_0}) \neq \rho_1(u_{j_0})$ . Without loss of generality suppose that  $\rho_0(u_{j_0}) = 0$  then  $\rho_1(u_{j_0}) = 1$ . We define a substitution  $\rho$  as follows:  $\rho(w_j) = \rho_0(u_j)$ .

- On the one hand substitution  $\rho_0 \circ \rho$  does not falsify  $\varphi$  since the only clauses that can be falsified are clauses associated with  $\{w_j \mid j \in [d]\}$  (we have already checked that the other clauses are satisfied). But the clause corresponding to the node  $w_j$  for  $\{j \in [d]\}$  is falsified iff 0 is substituted into  $w_j$  and 1 are substituted to all its immediate predecessors. But if  $\rho(w_j) = 0$  then  $\rho_0(u_j) = 0$ . Hence  $\varphi|_{\rho \circ \rho_0} = 1$ .
- On the other hand,  $\rho(w_{j_0}) = 0$ . The node  $w_{j_0}$  has two immediate predecessors: the node  $r_{j_0}$ , into which we always assign 1, and  $u_{j_0}$  such that  $\rho_1(u_{j_0}) = 1$ . Hence  $(u_{j_0}, r_{j_0} \rightarrow w_{j_0})|_{\rho \circ \rho_1} = 0$  and  $\varphi|_{\rho \circ \rho_1} = 0$ .

**Case 2:** Suppose that  $\{u_j \mid j \in [d]\}$  lies in the second part of the variable order.

Similarly to the Case 1, we define  $2^d$  substitutions into the variables  $\{w_j \mid j \in [d]\}$ . We need to show that substitutions  $\rho_0$  and  $\rho_1$  lead to different Boolean functions. Again we find  $j_0 \in [d]$  such that  $\rho_0(w_{j_0}) = 0$  and  $\rho_1(w_{j_0}) = 1$ . In this case we define  $\rho(u_j) = \rho_1(w_j)$ .

Similarly to the Case 1,  $\rho$  satisfies clauses for nodes from  $\{u_j \mid j \in [d]\}$ . Also  $\rho \circ \rho_1$  satisfy clauses for nodes from  $\{w_j \mid j \in [d]\}$  (we just copy  $\rho_1$  from  $\{w_j \mid j \in [d]\}$  to  $\{u_j \mid j \in [d]\}$ ). At the same time  $\rho(u_{j_0}) = 1$ ,  $r_{j_0}$  is always substituted with 1 and  $\rho_0(w_{j_0}) = 0$ . Hence  $(u_{j_0}, r_{j_0} \rightarrow w_{j_0})|_{\rho \circ \rho_0} = 0$  and  $\varphi|_{\rho \circ \rho_0} = 0$ . ◀



■ **Figure 3** An example of a substitution to a ball's variables.

### 3.2 Proof of Theorem 12

**Proof of Theorem 12.** Let us fix  $\alpha \in (0, 1)$ .

Now we show that for every OBDD( $\wedge$ , reordering) refutation of  $\text{Peb}(\text{Grid}_n)$  its depth is at least  $n^{1-\alpha}/2$  or its size is at least  $2^{n^\alpha/18928}$ .

Suppose there exists a refutation with depth less than  $n^{1-\alpha}/2$  and size less than  $2^{n^\alpha/18928}$ . Every refutation can be represented as directed acyclic graph such that:

- Each node is labeled with some OBDD from the refutation. Note that every such OBDD is equivalent to the conjunction of a subset of  $\text{Peb}(\text{Grid}_n)$  clauses. For every node, we add the conjunction to the label for clarity.
- Its only source is labeled with the constant false OBDD.
- Each sink is labeled with OBDD for some clause of  $\text{Peb}(\text{Grid}_n)$ .
- If an OBDD in a node is obtained by the conjunction rule then the node has outdegree 2 and the OBDD in the node is a conjunction of the OBDDs in the descendants.
- If an OBDD in a node is obtained by the reordering rule then the node has outdegree 1 and the OBDD in the node is the result of the reordering rule applied to the OBDD in the node's descendant.

Let us divide the subgrid  $[1, n - 1] \times [1, n - 1]$  into  $[1, n - n^\alpha] \times [1, n - n^\alpha]$ ,  $[n - n^\alpha + 1, n - 1] \times [n - n^\alpha + 1, n - 1]$ ,  $[n - n^\alpha + 1, n - 1] \times [1, n - n^\alpha]$ ,  $[1, n - n^\alpha] \times [n - n^\alpha + 1, n - 1]$  as in Subsection 3.1 (set  $m = n - 1$  and  $k = n^\alpha$ ).

Let  $\varphi$  be a CNF formula and let  $S$  be a subset of its clauses. We denote  $\varphi_S = \bigwedge_{C \in S} C$ .

Consider an arbitrary OBDD( $\wedge$ , reordering) refutation of  $\text{Peb}(\text{Grid}_n)$ . Consider its source. Since by Lemma 8,  $\text{Peb}(\text{Grid}_n)$  is minimal unsatisfiable, the source is labeled with the conjunction of all clauses (i.e.  $\text{Peb}(\text{Grid}_n)$  itself). In particular, all clauses for vertices from  $[1, m] \times [1, m]$  lie there. We start a path at the source of the refutation. If the current node (initially the current node is the source) has only one immediate descendant then we move into the descendant until the current node has two of them. Assume that the current node is labeled with  $\pi$ -OBDD for some order of the variables  $\pi$ . By the definition of the conjunction rule the current node's descendants are also labeled with  $\pi$ -OBDDs. For this order  $\pi$  and parameters  $m$  and  $k$  find sets  $U_\pi, W_\pi$  as was described in Subsection 3.1. Suppose one of the descendants is labeled with formula  $\varphi_{S_1}$  and the other with  $\varphi_{S_2}$ . Then

$[1, m] \times [1, m] \subset S_1 \cup S_2$ . Hence  $|S_1 \cap W_\pi| \geq |W_\pi|/2$  or  $|S_2 \cap W_\pi| \geq |W_\pi|/2$ . Without loss of generality  $|S_1 \cap W_\pi| \geq |W_\pi|/2$ . Recall that  $|W_\pi| \geq k/9464$  so  $|S_1 \cap W_\pi| \geq k/18928$ . Consider two cases: whether  $[1, n - n^\alpha - 1] \times [1, n - n^\alpha - 1] \not\subset S_1$  or  $[1, n - n^\alpha - 1] \times [1, n - n^\alpha - 1] \subset S_1$ .

**Case 1:**  $[1, n - n^\alpha - 1] \times [1, n - n^\alpha - 1] \not\subset S_1$ . In this case we can apply Lemma 13 with  $d = |S_1 \cap W_\pi| \geq k/18928 = n^\alpha/18928$  and variable order  $\pi$ . Hence the size of  $\pi$ -OBDD for  $\varphi_{S_1}$  from the current node is at least  $2^{n^\alpha/18928}$ . Therefore the size of the refutation is at least  $2^{n^\alpha/18928}$ . Hence this case is impossible.

**Case 2:**  $[1, n - n^\alpha - 1] \times [1, n - n^\alpha - 1] \subset S_1$ . All clauses from  $[1, n - n^\alpha - 1] \times [1, n - n^\alpha - 1]$  are still in the conjunction.

We divide this square into 4 subsquares, same as we did with  $[1, n - 1] \times [1, n - 1]$ . Now we set  $m = n - n^\alpha - 1$ ,  $k = n^\alpha$  and repeat the actions for new values of  $m$  and  $k$ . Again we can move down at least one time in the refutation's graph so that at the current node there will be all clauses from the top left subsquare (this time it is  $[1, n - 2n^\alpha - 2] \times [1, n - 2n^\alpha - 2]$ ).

Again, we divide this subsquare into 4 subsubsquares ( $m = n - 2n^\alpha - 2$ ,  $k = n^\alpha$ ) and so on. Case 1 is always impossible since  $k$  is always equal to  $n^\alpha$  and we assumed that the size of the refutation is less than  $2^{n^\alpha/18928}$ . We can repeat the process  $\frac{n}{n^\alpha+1} \geq n^{1-\alpha}/2$  times. Every time we move down in the refutation's graph at least once, therefore its depth is at least  $n^{1-\alpha}/2$ . ◀

► **Corollary 15.** *Dag-like and tree-like OBDD( $\wedge$ ) and OBDD( $\wedge$ , reordering) proofs cannot be balanced i.e. there is no polynomial  $p$  such that for every unsatisfiable formula  $\varphi$  and for every its refutation  $w$  (dag-like or tree-like) of the size  $S$ , there exists a refutation  $w'$  (dag-like or tree-like respectively) of the size  $p(S)$  and of the depth  $O(\log(S))$ .*

**Proof.** By Lemma 5 Formula  $\text{Peb}(\text{Grid}_n)$  has tree-like OBDD( $\wedge$ ) refutation of the size  $O(n^4)$ . But  $\text{Peb}(\text{Grid}_n)$  cannot have OBDD( $\wedge$ , reordering) a refutation of the size  $\text{poly}(n)$  and of the depth  $O(\log \text{poly}(n)) = O(\log n)$  due to Theorem 12. Hence the proof systems are not balanced. ◀

---

## References

- 1 Albert Atserias, Maria Luisa Bonet, and Jordi Levy. On Chvatal rank and cutting planes proofs. *Electron. Colloquium Comput. Complex.*, TR03-041, 2003. [arXiv:TR03-041](#).
- 2 Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 – October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004. doi:10.1007/978-3-540-30201-8\_9.
- 3 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.*, 22:319–351, 2004. doi:10.1613/jair.1410.
- 4 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In Bernard Chazelle, editor, *Innovations in Computer Science – ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 401–416. Tsinghua University Press, 2011. URL: <http://conference.iis.tsinghua.edu.cn/ICS2011/content/papers/3.html>.
- 5 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *J. ACM*, 48(2):149–169, March 2001. doi:10.1145/375827.375835.
- 6 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

- 7 Sam Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. *Electron. Colloquium Comput. Complex.*, TR18-041, 2018. arXiv:TR18-041.
- 8 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 24(3):305–317, 2005. doi:10.1109/TCAD.2004.842808.
- 9 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.
- 10 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discret. Appl. Math.*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.
- 11 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(04):439–562, August 2006. doi:10.1090/s0273-0979-06-01126-8.
- 12 Dmitry Itsykson, Alexander Knop, Andrei E. Romashchenko, and Dmitry Sokolov. On obdd-based algorithms and proof systems that dynamically change the order of variables. *J. Symb. Log.*, 85(2):632–670, 2020. doi:10.1017/jsl.2019.53.
- 13 Christoph Meinel and Anna Slobodová. On the complexity of constructing optimal ordered binary decision diagrams. In Igor Prívvara, Branislav Rován, and Peter Ruzicka, editors, *Mathematical Foundations of Computer Science 1994, 19th International Symposium, MFCS'94, Kosice, Slovakia, August 22–26, 1994, Proceedings*, volume 841 of *Lecture Notes in Computer Science*, pages 515–524. Springer, 1994. doi:10.1007/3-540-58338-6\_98.
- 14 Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD – Foundations and Applications*. Springer, 1998. URL: <http://www.informatik.uni-trier.de/~7Emeinel/books/obddbook.html>.
- 15 Pavel Pudlák and Samuel R. Buss. How to lie without being (easily) convicted and the length of proofs in propositional calculus. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 1994. doi:10.1007/BFb0022253.
- 16 Alasdair Urquhart. The depth of resolution proofs. *Stud Logica*, 99(1-3):349–364, 2011. doi:10.1007/s11225-011-9356-9.

## A Proof of Lemma 5

► **Proposition 16.** *Let  $X = \{x_1, \dots, x_n\}$  be a set of Boolean variables and let  $Y \subset X$ . Let  $\varphi = \bigwedge_{y \in Y} y$ . Then there exists  $\pi$ -OBDD for  $\varphi$  of the size  $O(|Y|)$  for every order of variables  $\pi$ .*

**Proof.** We enumerate  $Y$  according to the order  $\pi$ :  $Y = \{y_1, \dots, y_{|Y|}\}$ . We define  $\pi$ -OBDD for  $\varphi$  as follows: there is the unique node  $y_i$  for every  $i \in [|Y|]$ . We identify node and its label. The node  $y_1$  is the source. For every  $i \in [|Y|]$ ,  $y_i$ 's outgoing edge labeled with 0 goes to the sink labeled with 0. If  $i < |Y|$  then  $y_i$ 's outgoing edge labeled with 1 goes to  $y_{i+1}$  otherwise it goes to the sink labeled with 1. It is easy to see that there is a unique path between the source and the sink, labeled with 1, and that all edges on the path are labeled with 1. Hence the  $\pi$ -OBDD we have defined represents  $\varphi$ . ◀

► **Lemma 5 ([7]).** *For every directed acyclic graph  $G$  and for every order of variables  $\pi$  formula  $\text{Peb}(G)$  has tree-like  $\pi$ -OBDD( $\wedge$ ) refutation of size  $O(|V|^2)$  and depth  $O(|V|)$ .*

**Proof.** Let  $\{A_1, \dots, A_{n_1}\}$  be the first-type clauses in topological sort order. Let  $B$  be an arbitrary second-type clause. Consider the following sequence of the CNF formulas:  $A_1, A_1 \wedge A_2, \dots, \bigwedge_{i=1}^{n_1} A_i, (\bigwedge_{i=1}^{n_1} A_i) \wedge B$ . Represent each of this formulas as  $\pi$ -OBDD. Then it is easy to see that the sequence of OBDDs is a tree-like  $\pi$ -OBDD( $\wedge$ ) refutation. We now prove that the refutation has the size  $O(|V|^2)$ . It consists of  $|V| + 1$  formulas. It is sufficient to prove that each formula has the size  $O(|V|)$ . We consider two cases:

**Case 1:** The formula is  $A_1 \wedge \dots \wedge A_i$  for some  $i \in [n_1]$ . We prove by the induction on  $i$  that  $A_1 \wedge \dots \wedge A_i \equiv v_1 \wedge \dots \wedge v_i$ , where  $\equiv$  stands for logical equivalence. **Base:**  $i = 1$  so  $v_1$  is a source and the corresponding clause is  $(v_1)$ . **Induction step:** Assume that  $A_1 \wedge \dots \wedge A_i \equiv v_1 \wedge \dots \wedge v_i$  then  $A_1 \wedge \dots \wedge A_i \wedge A_{i+1} \equiv v_1 \wedge \dots \wedge v_i \wedge A_{i+1}$ . Let  $A_{i+1} = (u_1, \dots, u_k \rightarrow v_{i+1})$  where  $u_1, \dots, u_k$  is the set of all immediate predecessors of  $v_{i+1}$ . Since the clauses  $\{A_i \mid i \in [n_1]\}$  appear in the topological sort order then the first-type clauses that correspond to the variables  $\{u_1, \dots, u_k\}$  lie in  $\{A_1, \dots, A_i\}$ . Then it is easy to see that  $v_1 \wedge \dots \wedge v_i \wedge (u_1, \dots, u_k \rightarrow v_{i+1}) \equiv v_1 \wedge \dots \wedge v_i \wedge v_{i+1}$ .

Proposition 16 implies that such formulas have OBDD representation of the size  $O(|V|)$ .

**Case 2:** The formula is  $(\bigwedge_{i=1}^{n_1} A_i) \wedge B$ . We already have proved that  $(\bigwedge_{i=1}^{n_1} A_i) \equiv \bigwedge_j v_j$  is the conjunction of the all nodes. This conjunction implies that the values of variables of all nodes equal 1. At the same time the clause  $B$  implies that the variable of the corresponding sink equals 0. Hence the formula is unsatisfiable and the corresponding OBDD is constant false. ◀

## B Proof of Lemma 8

► **Lemma 8 (Folklore).** *Let  $G$  be a directed acyclic graph with only one sink. Then  $\text{Peb}(G)$  minimal unsatisfiable i.e. a conjunction of every proper subset of its set of clauses is satisfiable.*

**Proof.** Fix some proper subset  $S$  of the set of all clauses of  $\text{Peb}(G)$ . We now prove that  $\bigwedge_{C \in S} C$  is satisfiable.

Consider two cases:

**Case 1:** There is no second type clause in  $S$ . Then each clause from  $S$  contains literal without negation. Then the assignment of all 1 is satisfiable.

**Case 2:** There is no first-type clause in  $S$ . Denote the corresponding variable by  $v$ . Denote by  $t$  the unique sink of the graph. Note that there is a path from  $v$  to  $t$  (otherwise there are at least two sinks in the graph). Then the assignment of 0 to the path's variables and 1 to the other variables is satisfiable. Indeed, the first-type clauses corresponding to the variables substituted with 1 are always satisfied. The second-type clause is satisfied since the sink is substituted with 0. The other variables substituted with 0 have at least one immediate predecessor substituted with 0, hence their first type clauses are also satisfied. ◀



# Lower Bounds for Choiceless Polynomial Time via Symmetric XOR-Circuits

Benedikt Pago  

Mathematical Foundations of Computer Science, RWTH Aachen University, Germany

---

## Abstract

Choiceless Polynomial Time (CPT) is one of the few remaining candidate logics for capturing PTIME. In this paper, we make progress towards separating CPT from polynomial time by firstly establishing a connection between the expressive power of CPT and the existence of certain symmetric circuit families, and secondly, proving lower bounds against these circuits. We focus on the isomorphism problem of unordered Cai-Fürer-Immerman-graphs (the CFI-query) as a potential candidate for separating CPT from PTIME. Results by Dawar, Richerby and Rossman, and subsequently by Pakusa, Schalhöfer and Selman show that the CFI-query is CPT-definable on linearly ordered and preordered base graphs with small colour classes. We define a class of CPT-algorithms, that we call “CFI-symmetric algorithms”, which generalises all the known ones, and show that such algorithms can only define the CFI-query on a given class of base graphs if there exists a family of symmetric XOR-circuits with certain properties. These properties include that the circuits have the same symmetries as the base graphs, are of polynomial size, and satisfy certain fan-in restrictions. Then we prove that such circuits with slightly strengthened requirements (i.e. stronger symmetry and fan-in and fan-out restrictions) do not exist for the  $n$ -dimensional hypercubes as base graphs. This almost separates the CFI-symmetric algorithms from PTIME – up to the gap that remains between the circuits whose existence we can currently disprove and the circuits whose existence is necessary for the definability of the CFI-query by a CFI-symmetric algorithm.

**2012 ACM Subject Classification** Theory of computation → Finite Model Theory

**Keywords and phrases** logic in computer science, finite model theory, descriptive complexity, symmetric computation, symmetric circuits, graph isomorphism

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.73

**Related Version** *Full Version:* <https://arxiv.org/abs/2302.05426> [27]

**Acknowledgements** I thank Daniel Wiebking for his group-theoretic help.

## 1 Introduction

A central open question in finite model theory is whether there exists a logic that captures the complexity class polynomial time. It was first raised by Chandra and Harel in 1980 [5] and later made precise by Gurevich [20]. According to his definition, the term “logic” refers to any computation model that operates on finite structures and is *isomorphism-invariant*, i.e. yields the same output on isomorphic input structures. The question for a logic for PTIME thus asks whether there exists some isomorphism-invariant computation model which can decide exactly the same classes of structures that can be decided by “classical” polynomial time algorithms (i.e. Turing machines). The latter are not isomorphism-invariant because every structure, for example a graph, has multiple different representations as a binary string (e.g. depending on the vertex order that is used for the adjacency list/matrix). The result of a classical computation depends on this string representation and not on the isomorphism-type of the graph, which is undesirable for a logic. Another way of phrasing the question is whether the time and space cost of ensuring symmetry-invariance in computations is necessarily super-polynomial or not. Gurevich himself conjectured that no logic for PTIME exists, and if we had a proof for this, it would immediately separate P and NP: It is long known by Fagin’s



© Benedikt Pago;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 73; pp. 73:1–73:15

Leibniz International Proceedings in Informatics



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

theorem [14] that NP is captured by existential second order logic. In fact, many candidate logics that have been proposed in an attempt to capture PTIME have been proven to be only a strict fragment of it. For a survey on this topic (excluding the results from recent years), see [18].

One prominent logic for which research on lower bounds has not been so successful yet is *Choiceless Polynomial Time* (CPT). It was introduced in 1999 by Blass, Gurevich and Shelah [3] as a symmetry-invariant machine model. It has been open since then whether CPT (with counting) does capture PTIME or not. For more background information on CPT, see for example [17, 28, 32]. CPT can also be viewed as an extension of *fixed-point logic with counting* [6] with *hereditarily finite sets* as data structures. Other (perhaps less studied) candidates that have not been separated from PTIME so far are logics with witnessed choice constructs (a concept first introduced in [15]). These include fixed-point logic with witnessed symmetric choice and interpretations [23] and CPT extended with witnessed symmetric choice [24]. Prior to Lichter’s breakthrough [22], which separates *rank logic* from PTIME using a variation of the famous Cai-Fürer-Immerman (CFI) construction [4], logics with *linear-algebraic operators* [7] were also considered reasonable candidates. However, as outlined in [8], the results from [22] and [7] together imply that *no* set of isomorphism-invariant linear algebraic operators can be used to define a logic capturing PTIME. Thus, an important next step in this program would be to also rule out CPT as a logic for PTIME. In this article, we make progress towards this goal and propose an approach that allows to infer CPT lower bounds from lower bounds against certain symmetric Boolean circuit families. Thereby, the problem is narrowed down to the study of concrete combinatorial objects, for which we can present a first lower bound.

It should be remarked that the power of CPT is also of interest in another research context, namely with regards to the *graph isomorphism problem*: One can roughly divide the most important graph isomorphism algorithms into group-theoretic and combinatorial ones; the latter term refers to generalisations of the well-known *Weisfeiler-Leman* (WL) method, and these are choiceless. It is in a sense possible to characterise CPT as the class of all polynomial-time combinatorial graph isomorphism algorithms. A precise description of these has been given with the *Deep Weisfeiler-Leman* computation model [19], which was shown to be equivalent to CPT. Hence, lower bounds against CPT would also imply limitations for all combinatorial graph isomorphism algorithms.

Concerning CPT lower bounds, only relatively little is known. There is a non-definability result for a *functional* problem in PTIME, namely, it is impossible to define the dual of a given finite vector space in CPT [30]. What we would like to have is, however, the inexpressibility of a polynomial-time *decision problem*. We focus on a standard benchmark from finite model theory, namely the *CFI-query* [4]. Instances of this query are obtained by applying the so-called CFI construction to any connected undirected *base graph*, yielding either an “even” or an “odd” CFI-graph (see Section 3). The query asks to determine the parity of a given CFI-graph and it can be seen as a variant of the graph isomorphism problem or as the problem of solving a certain linear equation system over the finite field  $\mathbb{F}_2$ . Its descriptive complexity depends highly on the choice of base graphs and whether these come with a built-in linear order or not. The CFI-query is decidable in polynomial time but not in fixed-point logic with counting [4]. It is open whether it is CPT-definable on *unordered* base graphs, and our goal is to eventually answer this question in the negative. Our approach starts off from positive results: There do exist CPT-algorithms for linearly ordered and preordered versions of the CFI-query [9, 29] and also CFI-graphs over base graphs of linear degree [29]. In [26] it was shown that there exist unordered CFI-graphs (over  $n$ -dimensional hypercubes) whose degree is not linear and which cannot be preordered in CPT in such a way that the preorder-based algorithm from [29] (or the total-order-based one from [9]) could

be applied. This shows that these known choiceless algorithms for preordered versions of the CFI-query do not generalise to the unordered case because the necessary combinatorial objects (said preorders) are not symmetric enough. In the present paper, we define a general class of CPT-algorithms for the CFI-query, which encompasses all the known ones mentioned above, and show that their expressiveness depends on the existence of certain symmetric combinatorial objects, namely circuits with Boolean XOR-gates. We show that the CFI-query over a given class  $\mathcal{K}$  of base graphs is only definable by an algorithm from that class if there exists a family of polynomial-size symmetric XOR-circuits with the properties in Theorem 1. This means that the non-definability of the CFI-query over  $\mathcal{K}$  can be shown by proving the non-existence of such circuits. In Theorem 2, we almost achieve this goal: If we take as  $\mathcal{K}$  the family of  $n$ -dimensional hypercubes and make the circuit properties slightly more restrictive than required by our Theorem 1, then we succeed in showing that such circuits cannot exist. Thus, we come close to showing that the CFI-query over unordered hypercubes is undefinable by any CPT-algorithm from the class we are considering.

Following [9], we denote a CFI-graph over a base graph  $G$  or  $H$  as  $\mathfrak{G}^S$  or  $\mathfrak{H}^S$  (where  $S$  denotes the set of vertices whose CFI-gadget is odd). We consider circuits whose internal gates are XOR-gates and whose input gates are labelled with edges of the associated CFI base graph  $G$ . Therefore, the automorphism group of  $G$  has a natural action on the circuits as well. A circuit is said to be *sensitive* to a certain input bit if flipping just that bit changes the output. The other circuit properties will be explained in Section 5.

► **Theorem 1** (Main Theorem; see Theorem 31 in the full version for more details). *Let  $(G_n = (V_n, E_n))_{n \in \mathbb{N}}$  be a sequence of connected base graphs. Let  $\mathfrak{G}_n^S$  be a CFI-graph over  $G_n$ , and let  $\mathbf{tw}_n$  denote the treewidth of  $G_n$ . If there exists a CPT-program  $\Pi$  that is super-symmetric and CFI-symmetric and decides the CFI-query on all  $\mathfrak{G}_n^S$ , then there also exists a family  $(C_n)_{n \in \mathbb{N}}$  of XOR-circuits such that*

1. *The number of gates in  $C_n$  is polynomial in  $|\mathfrak{G}_n^S|$ .*
2. *The  $\mathbf{Aut}(G_n)$ -orbit of the circuit has size polynomial in  $|\mathfrak{G}_n^S|$ .*
3.  *$C_n$  is sensitive to  $\Omega(\mathbf{tw}_n)$  input bits.*
4. *The fan-in dimension of  $C_n$  is  $\mathcal{O}(\log |\mathfrak{G}_n^S|)$ .*

The terms *super-symmetric* and *CFI-symmetric* refer to the properties of a *hereditarily finite set* that is constructed by the program  $\Pi$  in order to decide the CFI-query. *Super-symmetry* is a property of h.f. sets that goes back to [9] (see Definition 5). *CFI-symmetry* is a concept that we define in this paper and which describes the internal structure and “local symmetries” of a h.f. set (see Section 4). The CFI-algorithms from [9] and [29] are based on super-symmetric and CFI-symmetric h.f. sets, and arguably, both these properties are crucial for the success of all these algorithms.

Our second main result shows that if we choose the  $n$ -dimensional hypercubes as the family of base graphs, and impose slightly stronger conditions on the circuits, then it is not possible to satisfy all of them together.

► **Theorem 2.** *Let  $(\mathcal{H}_n)_{n \in \mathbb{N}}$  be the family of  $n$ -dimensional hypercubes and let  $\mathbf{tw}_n$  denote the treewidth of  $\mathcal{H}_n$ . There exists no family of symmetric XOR-circuits  $(C_n)_{n \in \mathbb{N}}$  such that:*

1. *The number of gates in  $C_n$  is polynomial in  $|\mathfrak{H}_n^S|$ .*
2. *The  $\mathbf{Aut}(\mathcal{H}_n)$ -orbit of the circuit has size exactly one.*
3.  *$C_n$  is sensitive to  $\Omega(\mathbf{tw}_n)$  input bits.*
4. *For any two gates  $g, h$  in  $C_n$  such that  $h$  is a parent of  $g$ , it holds  $|\mathbf{Orbit}_{(h)}(g)| \in \mathcal{O}(\log |\mathfrak{H}_n^S|)$  and  $|\mathbf{Orbit}_{(g)}(h)| \in \mathcal{O}(\log |\mathfrak{H}_n^S|)$ .*

Here,  $\mathbf{Orbit}_{(h)}(g)$  denotes the orbit of the gate  $g$  with respect to the subgroup of  $\mathbf{Aut}(\mathcal{H}_n)$  that fixes the gate  $h$  (and vice versa for  $\mathbf{Orbit}_{(g)}(h)$ ).

If the four circuit properties were the same as in Theorem 1, then this would separate the class of super- and CFI-symmetric choiceless algorithms from PTIME. The difference between the two theorems is that here, the circuit has orbit size one, i.e. it is stabilised by the whole group  $\mathbf{Aut}(\mathcal{H}_n)$ , whereas in Theorem 1, the orbit of the circuit is only required to be polynomial. Moreover, here, we have a logarithmic bound on the parents and children (per orbit) of every gate, whereas in Theorem 1, the logarithmic bound is on the *fan-in dimension* of the gates. We define this notion in Section 5; we do not know if logarithmic fan-in dimension implies the orbit-wise logarithmic bound on the number of children (or vice versa), and probably, it does not imply the bound on the number of parents. So the gap between our two main results concerns how symmetric the circuits have to be and how restricted the connectivity between two consecutive circuit layers is. It remains a problem for future work to close this gap. This paper is thus a first step of a potentially longer programme towards showing  $\text{CPT} \neq \text{PTIME}$  via circuit lower bounds.

**Related work.** Lower bounds for symmetric circuits are studied in different contexts in the literature. For example, families of highly symmetric Boolean circuits with threshold gates characterise the power of fixed-point logic with counting [1], and certain more general circuits capture rank logic [11]. Our results are different in the sense that the XOR-circuits do not *characterise* CPT; they only represent the relevant structure of the h.f. sets that CPT uses to decide the CFI-query. Another line of research focuses on lower bounds for symmetric arithmetic circuits for the determinant and permanent polynomials, with the aim of making progress towards separating VP from VNP [10, 12]. Other examples of symmetric circuit lower bounds concern Boolean circuits for the parity function [31] and for computing products of permutation matrices [21]. The technical methods we employ in the proof of Theorem 2 might be applicable to the study of symmetric circuits in other contexts as well but we have not investigated this yet.

## 2 Choiceless Polynomial Time

**Hereditarily finite sets.** Let  $A$  be a finite set of atoms. The set of hereditarily finite objects over  $A$ ,  $\text{HF}(A)$ , is defined as  $\bigcup_{i \in \mathbb{N}} \text{HF}_i(A)$ , where  $\text{HF}_0(A) := A \cup \{\emptyset\}$ ,  $\text{HF}_{i+1}(A) := \text{HF}_i(A) \cup 2^{\text{HF}_i(A)}$ . The size of a h.f. set  $x \in \text{HF}(A)$  is measured in terms of its *transitive closure*  $\text{tc}(x)$ : The set  $\text{tc}(x)$  is the least transitive set such that  $x \in \text{tc}(x)$ . Transitivity means that for every  $a \in \text{tc}(x)$ ,  $a \subseteq \text{tc}(x)$ . Intuitively, one can view  $\text{tc}(x)$  as the set of all sets that appear as elements at some nesting depth within  $x$ .

**Choiceless Polynomial Time.** By CPT we always mean Choiceless Polynomial Time *with counting*. For details and various ways to define CPT formally, we refer to the literature: A concise survey can be found in [17]. The work by Blass, Gurevich and Shelah in which CPT was originally introduced as an abstract state machine model is [3]; later, more “logic-like” presentations of CPT were invented, such as Polynomial Interpretation Logic (see [16, 32]) and BGS-logic [30]. In short, CPT is like fixed-point logic with counting [6] plus a mechanism to construct isomorphism-invariant hereditarily finite sets of polynomial size. When a CPT-sentence (also called program)  $\Pi$  is evaluated in a finite structure  $\mathfrak{A}$ , then  $\Pi$  may augment  $\mathfrak{A}$  with hereditarily finite sets over its universe. The total number of distinct sets appearing in them (i.e. the sum over the sizes of the transitive closures of the h.f. sets) and the number of computation steps is bounded by  $p(|A|)$ , where  $p(n)$  is a polynomial that is explicitly part of the sentence  $\Pi$ . The run of  $\Pi$  on  $\mathfrak{A}$  is formally a sequence of computation stages, each of which is a h.f. set.

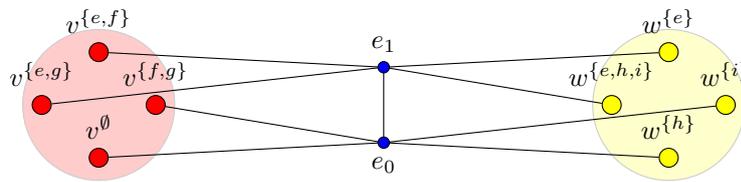
The h.f. sets that appear in the run of a program  $\Pi$  on a structure  $\mathfrak{A}$  are called the sets that are *activated* by  $\Pi$  on input  $\mathfrak{A}$ . Formally, the set of active objects is the union over the transitive closures of all the computation stages of the run of  $\Pi$  on  $\mathfrak{A}$ . The precise definition is not important for the purposes of this article and there exist multiple slightly varying definitions in the literature [9, 30, 32] which all essentially describe the same concept. The main limitation of CPT is that the set of objects activated by  $\Pi$  on  $\mathfrak{A}$  is closed under the automorphisms of  $\mathfrak{A}$  and at the same time of polynomial size in  $|A|$ . Thus, objects with super-polynomially large orbits cannot be activated by a CPT-program. It may be worth noting that while the whole set of activated objects has an orbit of size one, each activated object itself need not be fixed by *every* automorphism – as long as its orbit is only polynomial.

### 3 Unordered Cai-Fürer-Immerman graphs

Fix an undirected connected graph  $G = (V, E)$  as the *base graph* for the CFI-construction (whenever we speak of base graphs throughout the paper, we mean connected graphs). We turn  $G$  into a CFI-graph by replacing the edges with certain edge-gadgets and the vertices with vertex-gadgets. There are two types of vertex-gadgets, called odd and even. To construct a concrete CFI-graph over  $G$ , we have to fix a set  $S \subseteq V$  of vertices which are replaced by the *odd* gadget. The vertices in  $V \setminus S$  will be turned into the *even* gadget. We denote the resulting CFI-graph by  $\mathfrak{G}^S$ . The precise definition is as follows: Let  $\widehat{E} := \{e_0, e_1 \mid e \in E\}$ . These are the vertices that will form the edge-gadgets of  $\mathfrak{G}^S$ , so there are two vertices per edge-gadget. To define the vertices in vertex-gadgets, we let, for each  $v \in V$ ,  $v_S^* := \{v^X \mid X \subseteq E(v), |X| \text{ even}\}$  if  $v \notin S$ , and otherwise,  $v_S^* := \{v^X \mid X \subseteq E(v), |X| \text{ odd}\}$ . Here,  $E(v) \subseteq E$  are the edges incident to  $v$  in  $G$ . The vertices in  $v_S^*$  form the vertex-gadget of  $v$ . In total, we let  $\widehat{V}_S := \bigcup_{v \in V} v_S^*$ . Then the vertex-set of  $\mathfrak{G}^S$  is  $V(\mathfrak{G}^S) := \widehat{V}_S \cup \widehat{E}$ . The edges of the CFI-graph are given by

$$E(\mathfrak{G}^S) := \{\{v^X, e_i\} \mid v^X \in \widehat{V}_S, e_i \in \widehat{E}, |X \cap \{e\}| = i\} \cup \{\{e_0, e_1\} \mid e \in E\}.$$

In other words, for every  $v \in V$ , we connect each  $v^X \in v_S^*$  with the edge-gadgets of all edges  $e \in E(v)$  in such a way that  $v^X$  is connected with  $e_0$  if  $e \notin X$ , and otherwise with  $e_1$ . Also, we connect  $e_0$  and  $e_1$  to ensure that no automorphism of  $\mathfrak{G}^S$  can tear apart the edge-gadgets. Our CFI-graphs are unordered, so the only relation of the structure  $\mathfrak{G}^S$  is the edge relation  $E$ . The *CFI-query* asks for the parity of  $|S|$ , given a CFI-graph  $\mathfrak{G}^S$ . This is essentially the



■ **Figure 1** Gadgets  $v_S^*, w_S^*$  for  $v \notin S, w \in S$ , connected by the gadget for the edge  $e$ .

same question as the graph isomorphism problem for CFI-graphs:

► **Theorem 3** ([4, 9]). *For two given CFI-graphs over the same base graph, it holds  $\mathfrak{G}^S \cong \mathfrak{G}^R$  if and only if  $|S| \equiv |R| \pmod{2}$ .*

Alternatively, deciding the parity of  $|S|$  can be phrased as a linear equation system over  $\mathbb{F}_2$  in the variables  $\widehat{E}$  [2]. Since the reduction to a linear equation system is easily computable from the given CFI-graph  $\mathfrak{G}^S$ , and linear equation systems can be efficiently solved using, for example, Gaussian elimination, the CFI-query is decidable in polynomial time.

For logics that lack the ability to create higher-order objects, such as bounded-variable counting logic  $\mathcal{C}^k$  (and hence fixed-point logic with counting), it is provably impossible to distinguish non-isomorphic CFI-graphs, provided that the treewidth of the base graphs is super-constant:

► **Theorem 4** ([4, 2]). *Let  $G = (V, E)$  be an undirected connected graph with treewidth  $t$ . Then for any two sets  $S, S' \subseteq V$ , it holds  $\mathfrak{G}^S \equiv_{\mathcal{C}^t} \mathfrak{G}^{S'}$ , even if  $\mathfrak{G}^S \not\cong \mathfrak{G}^{S'}$ .*

### 3.1 Automorphisms of unordered CFI-graphs

For a CFI-graph  $\mathfrak{G}^S$  over an *unordered* base graph  $G = (V, E)$ , two different kinds of automorphisms play a role: Firstly, there are what we call “CFI-automorphisms”. These are induced by swapping  $e_0$  and  $e_1$  in some edge-gadgets (this is called “flipping the edge”). Secondly, there are the automorphisms of the underlying graph  $G$  itself.

To speak about the CFI-automorphisms, we use the terminology from [9]: For a given base graph  $G$ , we consider not only a concrete CFI-instance with odd and even vertex gadgets, but we can also construct the “full” CFI-graph  $\mathfrak{G}$ , in which every vertex gadget is both even and odd. Formally, for  $v \in V$ , let  $v^* := v_0^* \cup v_{\{v\}}^* = \{v^X \mid X \subseteq E(v)\}$ , and  $\widehat{V} := \bigcup_{v \in V} v^*$ . The vertex-set of  $\mathfrak{G}$  is  $\widehat{V} \cup \widehat{E}$ , and the edge-set is

$$E(\mathfrak{G}) := \{\{v^X, e_i\} \mid v^X \in \widehat{V}, e_i \in \widehat{E}, |X \cap \{e\}| = i\} \cup \{\{e_0, e_1\} \mid e \in E\}.$$

Every CFI-instance  $\mathfrak{G}^S$  is an induced subgraph of  $\mathfrak{G}$ . Some of the CFI-automorphisms of  $\mathfrak{G}$  are also automorphisms of  $\mathfrak{G}^S$ , but not all of them are. The other automorphisms of  $\mathfrak{G}$  induce isomorphisms from  $\mathfrak{G}^S$  into another isomorphic CFI-graph. For each edge  $e = \{v, w\} \in E$ , let  $\rho_e$  denote the automorphism of  $\mathfrak{G}$  induced by flipping the edge  $e$ . Formally,  $\rho_e(e_0) = e_1, \rho_e(e_1) = e_0$ , and  $\rho_e(v^X) = v^{X \Delta \{e\}}, \rho_e(w^X) = w^{X \Delta \{e\}}$  for all  $v^X, w^X \in v^* \cup w^*$ . All other vertices in  $\widehat{V}$  are fixed by  $\rho_e$ . One can check that this is indeed an automorphism of  $\mathfrak{G}$ ; furthermore,  $\rho_e$  is an *isomorphism* from any CFI-instance  $\mathfrak{G}^S$  to  $\mathfrak{G}^{S \Delta \{v, w\}}$  (see also [9]). It is easy to see that these edge-flip automorphisms commute, so for  $F = \{e^1, \dots, e^m\} \subseteq E$  we may write  $\rho_F$  for  $\rho_{e^1} \circ \rho_{e^2} \circ \dots \circ \rho_{e^m}$ . So in total, for every  $F \subseteq E$ ,  $\rho_F$  is an automorphism of  $\mathfrak{G}$ . If every  $v \in V$  is incident to an even number of edges in  $F$ , then  $\rho_F$  is also an automorphism of  $\mathfrak{G}^S$ , not only of  $\mathfrak{G}$ . To sum up, we have the following groups of CFI-automorphisms of  $\mathfrak{G}$  and  $\mathfrak{G}^S$ :  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G}) := \{\rho_F \mid F \subseteq E\}$ . This group is isomorphic to the Boolean vector space  $\mathbb{F}_2^E$ : Each  $F \subseteq E$  is identified with its characteristic vector  $\chi(F) \in \mathbb{F}_2^E$ . It holds  $\rho_F \circ \rho_{F'} = \rho_{F \Delta F'}$ , and this corresponds to the vector  $\chi(F) + \chi(F') \in \mathbb{F}_2^E$ . As already said, for a CFI-instance  $\mathfrak{G}^S$ , i.e. an induced subgraph of  $\mathfrak{G}$ , we have that  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G}^S)$  is isomorphic to a subspace of  $\mathbb{F}_2^E$ . In addition to the CFI-automorphisms, we also have to consider  $\mathbf{Aut}(G) \leq \mathbf{Sym}(V)$ , i.e. the automorphism group of the unordered base graph. In total, the automorphism group of the full CFI-graph  $\mathfrak{G}$  is isomorphic to the following semi-direct product:  $\mathbf{Aut}(\mathfrak{G}) \cong \mathbf{Aut}_{\text{CFI}}(\mathfrak{G}) \rtimes \mathbf{Aut}(G) = \{(\rho_F, \pi) \mid \rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G}), \pi \in \mathbf{Aut}(G)\}$ . The automorphism group  $\mathbf{Aut}(\mathfrak{G}^S)$  of a concrete CFI-instance is  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G}^S) \rtimes \mathbf{Aut}(G) \leq \mathbf{Aut}(\mathfrak{G})$ .

Sets that are CPT-definable in  $\mathfrak{G}^S$  only have to be symmetric with respect to the latter, but nonetheless, we also consider the full group  $\mathbf{Aut}(\mathfrak{G})$  because it simplifies the analysis. The sets we call *super-symmetric* (see below) are also  $\mathbf{Aut}(\mathfrak{G})$ -symmetric.

### 3.2 Symmetries and supports of hereditarily finite sets over CFI-graphs

Let  $\mathfrak{G}^S$  be a CFI-graph over  $G = (V, E)$  and  $x \in \text{HF}(\widehat{E})$ . All the groups from the previous section act on  $\widehat{E}$  and therefore also on  $\text{HF}(\widehat{E})$ . The action of any permutation  $\pi$  on a set  $x \in \text{HF}(\widehat{E})$  is given by  $\pi(x) = \{\pi(y) \mid y \in x\}$ . If  $x$  is an atom  $e_i$ , with  $i \in \{0, 1\}$ ,

$e = \{u, v\} \in E$ , and  $\pi \in \mathbf{Aut}(G)$ , then  $\pi(x) = \pi(e)_i$ , where  $\pi(e) = \{\pi(u), \pi(v)\} \in E$ . If  $\pi = \rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ , then  $\pi(e_i) = e_j$ , where  $j = i + |F \cap \{e\}| \pmod{2}$ . A permutation  $\pi$  stabilises an object  $x \in \text{HF}(\widehat{E})$ , if  $\pi(x) = x$ . As already said,  $\mathbf{Aut}(\mathfrak{G}^S)$  is composed of edge flips and automorphisms of the base graph. We separate the effect of these two subgroups on the elements of  $\text{HF}(\widehat{E})$  and consider the following orbits and stabilisers for  $x \in \text{HF}(\widehat{E})$ . The different orbits we consider are  $\mathbf{Orb}_E(x) := \{\rho_F(x) \mid \rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})\}$ ,  $\mathbf{Orb}_G(x) := \{\pi(x) \mid \pi \in \mathbf{Aut}(G)\}$ , and the corresponding stabilisers are  $\mathbf{Stab}_E(x) = \{\chi(F) \mid \rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G}), \rho_F(x) = x\}$  and  $\mathbf{Stab}_G(x) = \{\pi \in \mathbf{Aut}(G) \mid \pi(x) = x\}$ . We always view  $\mathbf{Stab}_E(x)$  as a subspace of the Boolean vector space  $\mathbb{F}_2^E$ . Furthermore, let  $\mathbf{maxOrb}_E(x) := \max_{y \in \text{tc}(x)} |\mathbf{Orb}_E(y)|$ .

In [9], the term *super-symmetry* was introduced for h.f. sets which are fixed by *all* automorphisms in  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ . Here, we use a slightly relaxed notion:

► **Definition 5** (Super-symmetric objects). *Fix a family of CFI-graphs  $(\mathfrak{G}_n^S)_{n \in \mathbb{N}}$  and a  $\mu_n \in \text{HF}(\widehat{E}_n)$  for every  $n$ . The objects  $\mu_n$  are super-symmetric if there exists a polynomial  $p$  such that  $|\mathbf{Orb}_E(\mu_n)| \leq p(|\mathfrak{G}_n^S|)$ .*

### Supports for CFI-automorphisms

Generally, a *support* of a permutation group  $\Gamma \leq \mathbf{Sym}(A)$  is a subset  $S \subseteq A$  such that the pointwise stabiliser of  $S$  in  $\mathbf{Sym}(A)$  is a subgroup of  $\Gamma$ . A support of a h.f. set is a support of its stabiliser group. For subgroups of  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ , we will use a different notion, that we call *CFI-support*. The reason why we need a specific type of support for these groups is because otherwise, the group  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$  does not admit unique minimum supports.

► **Definition 6** (CFI-support). *A CFI-support of an object  $x \in \text{HF}(\widehat{E})$  is a subset  $S \subseteq E$  such that every  $\rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$  with  $F \cap S = \emptyset$  fixes  $x$ .*

For the proof of the next lemma, we refer to the long version; it is not very difficult and similar to the proof of Lemma 26 in [3]. The lemma entails that every  $x \in \text{HF}(\widehat{E})$  has a unique smallest CFI-support.

► **Lemma 7.** *Let  $x \in \text{HF}(\widehat{E})$ . Let  $A_1, A_2 \subseteq E$  be CFI-supports of  $x$ . Then  $A_1 \cap A_2$  is also a CFI-support of  $x$ .*

► **Definition 8** (Minimal CFI-support). *For  $x \in \text{HF}(\widehat{E})$ ,  $\text{sup}_{\text{CFI}}(x) \subseteq E$  denotes the unique minimal subset of  $E$  that is a CFI-support of  $x$ .*

## 4 CFI-symmetric hereditarily finite sets and algorithms

The CFI-query is definable in CPT on instances that arise from linearly ordered base graphs, base graphs that come with a preorder with colour classes of logarithmic size, and base graphs of linear degree [9, 29]. All these CPT-algorithms depend on the construction of a particular super-symmetric h.f. set  $\mu \in \text{HF}(\widehat{E})$  that encodes the parity of  $|S|$ , given an instance  $\mathfrak{G}^S$ . We isolate another property of these h.f. sets, besides super-symmetry, which is responsible for their small orbit size and suitability for encoding parities. We call this *CFI-symmetry*. Intuitively, a set  $\mu \in \text{HF}(\widehat{E})$  is CFI-symmetric if its “building blocks” behave similarly as CFI-gadgets in CFI-graphs, in the sense that they are “flipped” whenever an even number of “incident gadgets” is flipped. These building blocks are the *connected components* of sets. To define these, let a CFI-graph  $\mathfrak{G}^S$  and a set  $\mu \in \text{HF}(\widehat{E})$  be fixed, and let  $\sim_E$  be the following equivalence relation on the elements  $x \in \text{tc}(\mu)$ : For  $x, x' \in \text{tc}(\mu)$ , we write  $x \sim_E x'$

iff there exists an edge-flip  $\rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$  such that  $x' = \rho_F(x)$ . The  $\sim_E$ -equivalence class in  $\text{tc}(\mu)$  of an object  $x \in \text{tc}(\mu)$  is denoted  $[x]$ . The relation  $\sim_E$  induces a partition  $\mathcal{C}(x)$  on each  $x \in \text{tc}(\mu)$ , namely  $\mathcal{C}(x) := \{([y] \cap x) \mid y \in x\}$ . In [9], the elements of  $\mathcal{C}(x)$  are called the *connected components* of  $x$ . Now in a *CFI-symmetric* object, each connected component  $\gamma \in \mathcal{C}(x)$ , for each  $x \in \text{tc}(\mu)$ , behaves like a CFI-gadget. That is, the component has exactly two images under  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ , namely itself and its “flip”. Consider the following example of a small “parity-tracking” h.f. set that is constructed similarly as in the algorithms from [9] and [29].

► **Example 9.** Here is an example h.f. set  $\mu_{\{e,f,g\}} \in \text{HF}(\widehat{E})$  with  $E = \{e, f, g\}$ . It tracks the parity of edge-flips for the edges  $e, f, g$ . For better readability, the set is printed in a structured form, so the sets  $\mu_{\{f,g\}}$  and  $\tilde{\mu}_{\{f,g\}}$  are shown in the level below. Each of the

$$\mu_{\{e,f,g\}} = \left\{ \begin{array}{l} \mu_{\{f,g\}}, e_0 \\ \tilde{\mu}_{\{f,g\}}, e_1 \end{array} \right\}$$

$\{\{f_0, g_0\}, \{f_1, g_1\}\}$

$\{\{f_0, g_1\}, \{f_1, g_0\}\}$

$\mu$ -objects has only one connected component that consists of two sets which are related by  $\sim_E$ . For example, the set  $\mu_{\{e,f,g\}}$  is stabilised setwise whenever an even number of edges is flipped. The two elements of  $\mu_{\{e,f,g\}}$  themselves have two connected components: Clearly,  $e_0$  and  $\mu_{\{f,g\}}$  cannot be mapped to each other by any edge-flip. The same goes for example for  $f_0$  and  $g_0$ . They form distinct components of the set  $\{f_0, g_0\}$ , while  $\{\{f_0, g_0\}, \{f_1, g_1\}\}$  again only has one component that is stabilised if and only if an even number of edges in  $\{f, g\}$  is flipped. This pattern of alternation between sets with two components and sets with one component is typical of the super-symmetric objects constructed by the known CFI-algorithms.

► **Definition 10** (CFI-symmetric components and objects). *Let  $\mu \in \text{HF}(\widehat{E})$ ,  $x \in \text{tc}(\mu)$ , and  $\gamma \subseteq x$  be a connected component of  $x$ . Then we say that  $\gamma$  is CFI-symmetric if  $|\text{Orb}_E(\gamma)| = 2$ , and for each  $\rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ , it holds that  $\rho_F(\gamma) = \gamma$  iff for one/every  $y \in \gamma$ , the number of flipped components of  $y$ , that is  $|\{\gamma' \in \mathcal{C}(y) \mid \rho_F(\gamma') \neq \gamma'\}|$ , is even.*

*The set  $\mu$  is CFI-symmetric if the following two conditions are satisfied:*

1. *For each  $\rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ , it holds that  $\rho_F(\mu) = \mu$  iff the number of flipped components of  $\mu$ , that is,  $|\{\gamma \in \mathcal{C}(\mu) \mid \rho_F(\gamma) \neq \gamma\}|$ , is even.*
2. *For every  $x \in \text{tc}(\mu)$ , every connected component  $\gamma \in \mathcal{C}(x)$  is CFI-symmetric.*

In the full version, we show that the formulation “one/every” in the above definition is indeed justified. We call a CPT-program  $\Pi$  that decides the CFI-query on a class  $\mathcal{K}$  of base graphs *CFI-symmetric* if it activates a CFI-symmetric h.f. set  $\mu \in \text{HF}(\widehat{E})$  on every input  $\mathfrak{G}^S$  over every base graph  $G \in \mathcal{K}$ . To be precise,  $\mu$  must also have sufficient support size in order to enable the program to decide the CFI-query. This support lower bound is stated in Theorem 16. Similarly, we say that  $\Pi$  is *super-symmetric* if it activates a super-symmetric object of sufficient support.

## 5 Translating hereditarily finite sets to XOR-circuits

An XOR-circuit is a connected directed acyclic graph  $C = (V_C, E_C)$  with a unique designated root  $r$ . Its internal nodes are understood as XOR-gates and its leaves correspond to the input gates of the circuit. If  $(g, h) \in E_C$ , then the output of gate  $h$  is an input of gate  $g$ . Every XOR-circuit computes the Boolean XOR-function over a subset of its input bits.

We say that an XOR-circuit  $C$  is a circuit *over a graph*  $G = (V, E)$ , if the input gates of  $C$  are labelled with the edges in  $E$ . More precisely, let  $L \subseteq V_C$  be the leaves of  $C$ . There is an injective labelling function  $\ell : L \rightarrow E$  that relates the input gates with edges of  $G$ . To speak about the semantics of the circuit, we introduce a set of formal propositional variables  $\mathcal{V}(G) := \{X_e \mid e \in E\}$ . Every input gate  $g \in L$  is associated with the formal variable  $X_{\ell(g)}$ . Since every internal gate is an XOR-gate, the function computed by it is the XOR over a subset of  $\mathcal{V}(G)$ . For our purposes, this variable set (or actually the set of associated edges) is the main interesting property of a gate, and we call it  $\mathcal{X}(g)$ . Formally, if  $g \in L$ , then  $\mathcal{X}(g) := \{\ell(g)\} \subseteq E$ . If  $g$  is an internal gate, then  $\mathcal{X}(g) := \Delta_{h \in gE_C} \mathcal{X}(h)$ , that is, the symmetric difference over the  $\mathcal{X}(h)$  for all children of  $g$ . In other words,  $\mathcal{X}(g) \subseteq E$  is precisely the set of edges in  $E$  such that  $g$  computes the Boolean function  $\bigoplus_{e \in \mathcal{X}(g)} X_e$ . Thus, a gate  $g$  is *sensitive* to an input bit  $X_e$  if and only if  $e \in \mathcal{X}(g)$ . The function computed by the circuit  $C$  is the XOR over  $\mathcal{X}(r) \subseteq E$ , where  $r$  is the root of  $C$ .

## 5.1 Symmetries of circuits

A circuit  $C$  over a graph  $G$  is subject to the action of the automorphism group  $\mathbf{Aut}(G) \leq \mathbf{Sym}(V)$ . Any  $\pi \in \mathbf{Aut}(G)$  changes the labels of the input gates in  $L$ . So let  $g \in L$  with  $\ell(g) = e \in E$ . Then  $\pi(g)$  is an input gate with  $\ell(\pi(g)) = \pi(e)$ . The circuit  $\pi(C)$  is just  $C$  with the input labels modified accordingly. We say that  $\pi$  *extends to an automorphism* of  $C$  if there exists a bijection  $\sigma : V_C \rightarrow V_C$  that is an automorphism of the graph  $(V_C, E_C)$  and satisfies for each input gate  $g \in V_C$ :  $\ell(\sigma(g)) = \pi(\ell(g))$ . We write  $\mathbf{Stab}_G(C) = \{\pi \in \mathbf{Aut}(G) \mid \pi \text{ extends to an automorphism of } C\} \leq \mathbf{Aut}(G)$ , and  $\mathbf{Orb}_G(C) = \{\pi(C) \mid \pi \in \mathbf{Aut}(G)\}$ .

## 5.2 The parameter fan-in dimension

The XOR-circuits we will construct from CFI-symmetric h.f. sets will satisfy a certain fan-in bound on the gates. However, this bound will not be – as it is more common – on the number of incoming wires of a gate but rather, on the “linear algebraic complexity of incoming information”, so to say. The subsets of  $E$  form a Boolean vector space together with the symmetric difference operation. This space is isomorphic to  $\mathbb{F}_2^E$ .

For each gate  $g$  of an XOR-circuit  $C = (V_C, E_C)$ ,  $gE_C$  denotes the set of its children and  $E_C g$  the set of parents. With each internal gate  $g$  of an XOR-circuit  $C$  over a graph  $G = (V, E)$ , we can associate a Boolean matrix  $M(g) \in \mathbb{F}_2^{gE_C \times E}$ , that we call the *gate matrix*: The row at index  $h \in gE_C$  is defined as the characteristic vector of  $\mathcal{X}(h) \subseteq E$ , transposed, i.e.  $M(g)_{h-} = \chi(\mathcal{X}(h))^T$ . Here and in what follows, we write  $\chi$  for the bijection from  $\mathcal{P}(E)$  to  $\mathbb{F}_2^E$  that associates with each subset of  $E$  its characteristic Boolean vector. If  $g$  is an input gate, then we define  $M(g) \in \mathbb{F}_2^{[1] \times E}$  as the one-row matrix whose only row is  $\chi(\mathcal{X}(g))^T = \chi(\{\ell(g)\})^T$ .

The *fan-in dimension* of a gate  $g$  is the dimension of the row-space of  $M(g)$ ; this is the subspace of  $\mathbb{F}_2^E$  that is spanned by the characteristic vectors  $\chi(\mathcal{X}(h)) \in \mathbb{F}_2^E$ , for all children  $h$  of  $g$ . Equivalently, the fan-in dimension of  $g$  is  $\mathbf{rk}(M(g))$ . The *fan-in dimension* of the circuit  $C$  is the maximum fan-in dimension of any of its gates. The notion of fan-in dimension is unusual but as we will show, it nicely captures the orbit size of the original h.f. set with respect to the group of edge flips  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ . The  $\mathbf{Aut}(G)$ -symmetries of the h.f. set will be reflected in the symmetries of the circuit.

### 5.3 The circuit construction

► **Theorem 11.** Fix a family  $(G_n)_{n \in \mathbb{N}}$  of base graphs. For every  $n \in \mathbb{N}$ , let  $\mathfrak{G}_n^S$  be a CFI-graph over  $G_n = (V_n, E_n)$  and let  $\mu_n \in \text{HF}(\widehat{E}_n)$  be a CFI-symmetric h.f. set that is activated by a CPT-program on input  $\mathfrak{G}_n^S$  (by the same CPT-program for the whole family of graphs). Then for every  $n \in \mathbb{N}$ , there exists an XOR-circuit  $C(\mu_n) = (V_C, E_C)$  over the edges of  $G_n$  which satisfies:

1. The size of the circuit, i.e.  $|V_C|$ , is polynomial in  $|\mathfrak{G}_n^S|$ .
2. The orbit-size  $|\text{Orb}_G(C(\mu_n))|$  of the circuit is polynomial in  $|\mathfrak{G}_n^S|$ .
3.  $C(\mu_n)$  is sensitive to an edge  $e \in E_n$  if and only if  $e \in \text{sup}_{\text{CFI}}(\mu_n)$ .
4. The fan-in dimension of  $C(\mu)$  is  $\mathcal{O}(\log(\mathbf{maxOrb}_E(\mu)))$ . Recall that  $\mathbf{maxOrb}_E(\mu) = \max_{y \in \text{tc}(x)} |\text{Orb}_E(y)|$ .

For the proof, we fix  $\mu \in \text{HF}(\widehat{E})$  and denote by  $C(\mu) = (V_C, E_C)$  the corresponding XOR-circuit that we are going to define. The circuit is simply the factorised DAG-structure  $(\text{tc}(\mu), \in)_{/\sim_E}$ : The gates of the circuit are the  $\sim_E$ -equivalence classes (i.e.  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ -orbits) of the objects in  $\text{tc}(\mu)$ . Note that these orbits need not be subsets of  $\text{tc}(\mu)$ , so whenever we write  $[x]$ , we formally mean the orbit restricted to  $\text{tc}(\mu)$ :  $[x] = \{\rho_F(x) \mid \rho_F \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G}) \text{ such that } \rho_F(x) \in \text{tc}(\mu)\}$ . The circuit  $C(\mu)$  is defined as follows:

- $V_C := \text{tc}(\mu)_{/\sim_E} = \{[x] \mid x \in \text{tc}(\mu)\}$ .
- $E_C := \{([x], [y]) \mid \text{there exists } y' \in [y] \text{ such that } y' \in x\}$ .
- By definition, the leaves of  $C(\mu)$  correspond to  $\sim_E$ -classes of atoms in  $\text{tc}(\mu)$ . The set of atoms is  $\widehat{E}$ , so any leaf of  $C$  has the form  $[e_i]$ , for some  $e \in E, i \in \{0, 1\}$ . We let  $\ell([e_i]) := e$ .
- The root  $r$  of  $C(\mu)$  is  $[\mu]$ .

One can prove that the set of edges  $E_C$  can indeed be defined in this way: Whether or not there is an  $E_C$ -edge between  $[x]$  and  $[y]$  is independent of the choice of the representative of  $[x]$  in the definition. This is because all members of  $[x]$  are symmetric to each other in the DAG-structure  $(\text{tc}(\mu), \in)$ . Now we verify that the circuit has the desired properties. We start with the  $\mathbf{Aut}(G)$ -symmetry.

► **Lemma 12.** Every  $\pi \in \mathbf{Stab}_G(\mu) \leq \mathbf{Sym}(V)$  extends to an automorphism of the circuit  $C(\mu)$ , that is:  $\mathbf{Stab}_G(\mu) \leq \mathbf{Stab}_G(C(\mu))$ .

**Proof sketch.** Let  $\pi \in \mathbf{Stab}_G(\mu) \leq \mathbf{Aut}(G)$ . That is,  $\pi$  extends to an automorphism  $\sigma : \text{tc}(\mu) \rightarrow \text{tc}(\mu)$  of  $(\text{tc}(\mu), \in)$ . We define  $\sigma' : V_C \rightarrow V_C$  by letting  $\sigma'([x]) = [\sigma(x)]$ . This is well-defined because  $x \sim_E x'$  if and only if  $\sigma(x) \sim_E \sigma(x')$  ( $\sigma$  is an automorphism of  $\mu$ ). One can verify that  $\sigma'$  is an automorphism of  $C(\mu)$  that  $\pi$  extends to. ◀

► **Corollary 13.**  $|\text{Orb}_G(C(\mu))| \leq |\text{Orb}_G(\mu)|$ .

**Proof.** Follows from Lemma 12 together with the Orbit-Stabiliser Theorem, which says that  $|\text{Orb}_G(C(\mu))| = |\mathbf{Aut}(G)|/|\mathbf{Stab}_G(C(\mu))|$  and  $|\text{Orb}_G(\mu) = |\mathbf{Aut}(G)|/|\mathbf{Stab}_G(\mu)|$ . ◀

Next, we would like to analyse the fan-in dimension of  $C(\mu)$ , and the connection between  $C(\mu)$  and  $\text{sup}_{\text{CFI}}(\mu)$ . The key for this is to establish a connection between the stabilisers  $\mathbf{Stab}_E(x)$ , for all  $x \in \text{tc}(\mu)$ , and the kernels of the corresponding gate matrices. For the definition of these matrices, we refer back to Section 5.2.

► **Lemma 14.** For every gate  $[x] \in V_C$  and its gate matrix  $M[x] \in \mathbb{F}_2^{[x]E_C \times E}$ , it holds:  $\text{Ker}(M[x]) = \mathbf{Stab}_E(x) = \mathbf{Stab}_E(x')$  for every  $x' \in [x]$ . For every row  $M[x]_{[y]-}$ , for every  $[y] \in [x]E_C$ , it holds:

$$\text{Ker}(M[x]_{[y]-}) = \mathbf{Stab}_E([y] \cap x) \quad (\star)$$

**Proof.** It holds  $\mathbf{Stab}_E(x) = \mathbf{Stab}_E(x')$ , for every  $x' \in [x]$  and also  $\mathbf{Stab}_E([y] \cap x) = \mathbf{Stab}_E([y] \cap x')$ , for every  $x' \in [x]$  (because  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$  is Abelian). Therefore, equation  $(\star)$  does not depend on the choice of representatives. From  $(\star)$  it immediately follows that  $\mathbf{Ker}(M[x]) = \mathbf{Stab}_E(x)$ , because: The stabiliser of  $x$  is the intersection of the stabilisers of all connected components of  $x$ , and the kernel of  $M[x]$  is the intersection of the kernels of the individual rows of the matrix. We now prove  $(\star)$  via induction from the input gates to the root. If  $[x] = [e_0]$  is an input gate, then  $M[x]$  has just one row, which is  $\chi(e)^T$ . The kernel of  $\chi(e)^T$  is the set of all vectors in  $\mathbb{F}_2^E$  which are zero at index  $e$ . This is precisely  $\mathbf{Stab}_E(e_0) = \mathbf{Stab}_E(e_1)$ , as desired. Now suppose  $[x]$  is an internal gate, i.e.  $x$  is a non-atomic h.f. set in  $\text{tc}(\mu)$ . Each row of  $M[x] \in \mathbb{F}_2^{[x]E_C \times E}$  is the characteristic vector of  $\mathcal{X}[y] \subseteq E$ , for a  $[y] \in [x]E_C$ . Now fix such a child  $y$  of  $x$ . We have  $\mathcal{X}[y] = \Delta_{[w] \in [y]E_C} \mathcal{X}[w]$ . In matrix-vector notation, we can write this as:

$$M[x]_{[y]-} = \chi(\mathcal{X}([y]))^T = \sum_{[w] \in [y]E_C} (M[y]_{[w]-})^T = (1 \ 1 \ \dots \ 1) \cdot M[y].$$

Let  $\gamma \in \mathcal{C}(x)$  be the connected component such that  $\gamma = [y] \cap x$ . The equation above means that  $\mathbf{Ker}(M[x]_{[y]-}) = \mathcal{E}_y$ , where  $\mathcal{E}_y$  denotes the set of all vectors in  $\mathbb{F}_2^E$  whose image under  $M[y]$  has even Hamming weight. Thus we have to show that  $\mathcal{E}_y = \mathbf{Stab}_E(\gamma)$ . Each row  $M[y]_{[w]-}$  corresponds to a connected component  $\gamma' \in \mathcal{C}(y)$  with  $w \in \gamma'$ .

By the induction hypothesis, we have for each row  $M[y]_{[w]-}$  and each  $\mathbf{v} \in \mathbb{F}_2^E$  that  $M[y]_{[w]-} \cdot \mathbf{v} = 1$  iff  $\mathbf{v} \notin \mathbf{Stab}_E([w] \cap y)$ . So  $M[y] \cdot \mathbf{v}$  has even Hamming weight iff  $\rho_{\chi^{-1}(\mathbf{v})} \in \mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$  flips an even number of connected components of  $y$ . This is true iff  $\rho_{\chi^{-1}(\mathbf{v})}$  flips an even number of components in every  $y' \in \gamma$ . By definition of CFI-symmetry (Definition 10), this is the case iff  $\mathbf{v} \in \mathbf{Stab}_E(\gamma)$ , because  $\mu$  is CFI-symmetric, and thus,  $\gamma$  is a CFI-symmetric component. In total, we have shown that  $\mathbf{v} \in \mathcal{E}_y$  iff  $\mathbf{v} \in \mathbf{Stab}_E(\gamma)$ . This proves  $(\star)$  for every row of  $M[x]$ .  $\blacktriangleleft$

As a consequence of this correspondence between kernels and stabilisers, we can bound the fan-in dimension of  $C(\mu)$ . This proves **Property 4** from Theorem 11.

► **Lemma 15.** *The fan-in dimension of  $C(\mu)$  is  $\log(\mathbf{maxOrb}_E(\mu))$ .*

**Proof.** Let  $x \in \text{tc}(\mu)$ . From the Orbit-Stabiliser Theorem and the fact that  $|\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})| = 2^{|E|}$ , it follows that  $\mathbf{Orb}_E(x) = \frac{2^{|E|}}{|\mathbf{Stab}_E(x)|} \leq \mathbf{maxOrb}_E(\mu)$ . By Lemma 14,  $\mathbf{Stab}_E(x) = \mathbf{Ker}(M[x])$ . Applying the Rank Theorem to  $M[x]$ , we get:  $\mathbf{rk}(M[x]) = |E| - \dim \mathbf{Stab}_E(x) \leq \log(\mathbf{maxOrb}_E(\mu))$ . Since there is an object  $x \in \text{tc}(\mu)$  where  $\mathbf{maxOrb}_E(\mu)$  is attained,  $\mathbf{rk}(M[x]) = \log(\mathbf{maxOrb}_E(\mu))$  is indeed the maximum rank of any gate matrix of  $C(\mu)$ .  $\blacktriangleleft$

**Proof of Theorem 11.** First of all, since  $\mu$  is by assumption activated by a CPT-sentence in the structure  $\mathfrak{G}^S$ , the size  $|\text{tc}(\mu)|$  and the orbit  $|\mathbf{Orb}_{\mathbf{Aut}(\mathfrak{G}^S)}(\mu)|$  are polynomial in  $|\mathfrak{G}^S|$ . Therefore, **Property 1** from Theorem 11 clearly holds for  $C(\mu)$ , because  $|V_C| \leq |\text{tc}(\mu)|$ . **Property 2** follows from the bound on  $|\mathbf{Orb}_{\mathbf{Aut}(\mathfrak{G}^S)}(\mu)|$  together with Corollary 13, and the fact that  $|\mathbf{Orb}_G(\mu)| \leq |\mathbf{Orb}_{\mathbf{Aut}(\mathfrak{G}^S)}(\mu)|$ . **Property 4** is proven in Lemma 15. Finally, **Property 3** can be seen as follows: Suppose  $C(\mu)$  is sensitive to an edge  $e \in E$ . This means that  $e \in \mathcal{X}(r)$ , for the root  $r = [\mu]$  of  $C(\mu)$ . This is the case iff  $e \in \mathcal{X}[y]$  for an odd number of children  $[y] \in [\mu]E_C$ . This is the same as saying that the column  $M[\mu]_{-e}$  has odd Hamming weight. By equation  $(\star)$  from Lemma 14, this holds if and only if  $\chi(e) \notin \mathbf{Stab}_E([y] \cap x)$  for an odd number of children  $[y] \in [\mu]E_C$ . Since  $\mu$  is CFI-symmetric, by Definition 10 this is the case if and only if  $\rho_e(\mu) \neq \mu$ . And this holds iff  $e \in \text{sup}_{\text{CFI}}(\mu)$  (because  $\text{sup}_{\text{CFI}}(\mu)$  is the smallest possible CFI-support of  $\mu$ ).  $\blacktriangleleft$

## 5.4 Proving the main theorem

So far, we have a translation of CFI-symmetric h.f. sets in  $\text{HF}(\widehat{E})$  into XOR-circuits with the properties mentioned in Theorem 11. In order to conclude Theorem 1 from this, we additionally need the following: Any CPT-algorithm which is both super-symmetric and CFI-symmetric and decides the CFI-query must construct a h.f. set whose properties translate into the circuit properties from Theorem 1. Fortunately, a result to this effect exists already. The following support lower bound for general CPT-programs deciding the CFI-query is due to Dawar, Richerby, and Rossman [9].

► **Theorem 16** (implicit in the proof of Theorem 40 in [9]). *Let  $(G_n)_{n \in \mathbb{N}}$  be a family of base graphs and let  $tw_n$  denote the treewidth of  $G_n$ . Let  $\mathfrak{G}_n^0, \mathfrak{G}_n^1$  denote the even and odd CFI-structures over  $G_n$ . Assume that  $\mathfrak{G}_n^0$  and  $\mathfrak{G}_n^1$  are  $\mathcal{C}^{tw_n}$ -homogeneous. Then any CPT-program that distinguishes  $\mathfrak{G}_n^0$  and  $\mathfrak{G}_n^1$  for all  $n \in \mathbb{N}$  must activate on input  $\mathfrak{G}_n^i$  a h.f. set  $\mu_n$  whose smallest support has size at least  $\Omega(tw_n)$ .*

A structure  $\mathfrak{G}_n^S$  is  $\mathcal{C}^{f(n)}$ -homogeneous if whenever two tuples  $\bar{a}$  and  $\bar{b}$  satisfy exactly the same  $\mathcal{C}^{f(n)}$ -formulas in  $\mathfrak{G}_n^S$ , then there is an automorphism of  $\mathfrak{G}_n^S$  that maps  $\bar{a}$  to  $\bar{b}$ . In particular, this condition is satisfied by certain ordered CFI-graphs, as stated in [9], and one can also show that the unordered CFI-graphs over hypercubes, which we use for the lower bound in Theorem 2, satisfy it. There are other details which must be taken into account when connecting Theorem 16 with Theorem 11 in order to prove Theorem 1, e.g. one has to reconcile the different notions of support that these theorems talk about. We gloss over these things in this extended abstract and refer to Theorem 31 in the long version, which is the more precise formulation of Theorem 1. For our application to hypercube CFI-structures, it is shown in the long version that Theorem 1 really holds in this shortened formulation. The reason why Theorem 1 requires the object to be *super-symmetric* is that this allows us to infer that  $\mathcal{O}(\log \mathbf{maxOrb}_E(\mu)) \leq \mathcal{O}(\log |\mathfrak{G}_n^S|)$ . This is needed to translate Property 4 from Theorem 11 into Property 4 from Theorem 1. Super-symmetry together with the fact that  $|\text{tc}(\mu)|$  is polynomial guarantees that  $\mathbf{maxOrb}_E(\mu)$  is polynomially bounded in  $|\mathfrak{G}_n^S|$  (see Definition 5).

## 6 A lower bound for symmetric XOR-circuits

The detailed proof of Theorem 2 is too long for this extended abstract but here is an outline: The theorem states that no family of XOR-circuits exists which are stabilised by all automorphisms of the  $n$ -dimensional hypercube, are sensitive to sufficiently many input bits, of polynomial size, and satisfy certain logarithmic bounds on the orbit-wise number of parents and children of each gate. What the proof concretely shows is that the sensitivity requirement (condition 3 in Theorem 2) contradicts the other three conditions: Any sufficiently symmetric circuit  $C$  is so highly connected that most input bits cancel themselves out because the number of distinct (not necessarily vertex-disjoint) paths from the root  $r$  to the input is even, and every operation is XOR. To prove that this self-cancellation effect takes place for an input gate  $g$ , we partition the set of paths between  $g$  and  $r$  in  $C$  into their orbits. Then we show that each orbit contains an even number of paths. To achieve this, it suffices to look at one path  $P$  in each orbit and to show the existence of an edge  $(h', h)$  in this path such that  $\mathbf{Orbit}_{(h)}(h') := \{\pi(h') \mid \pi \in \mathbf{Stab}(h)\}$  has even size. Then we can conclude that  $P$  splits into an even number of “alternative routes” towards the root at gate  $h$ . Thus, the goal is to show that such an edge  $(h', h)$  exists on every path between  $r$  and  $g$ .

We do this by maintaining fine-grained information about the stabiliser groups  $\mathbf{Stab}(h_i)$  of the gates  $(h_1 = g, h_2, h_3, \dots, r)$  along a given path (the path here is presented in the reverse direction of the edges). The automorphism group of the  $n$ -dimensional hypercube contains as a subgroup the symmetric group  $\mathbf{Sym}_n$  acting on the  $n$  positions of the binary words  $\{0, 1\}^n$ , which form the vertex set of the hypercube. Thus, each  $\mathbf{Stab}(h_i)$  can be seen as a subgroup of  $\mathbf{Sym}_n$ . Such subgroups can be approximated by what we call their *coarsest alternating supporting partition*  $\mathbf{SP}_A(\mathbf{Stab}(h_i))$  (inspired by a similar concept used in [1]). This is the coarsest partition  $\mathcal{P}$  of  $[n]$  such that for each  $P \in \mathcal{P}$ , every even permutation of  $P$  is a member of  $\mathbf{Stab}(h_i)$ . We know what  $\mathbf{SP}_A(\mathbf{Stab}(g))$  looks like for every input gate  $g$ : This depends on the edge of the hypercube that  $g$  is labelled with. For a large proportion of the hypercube edges, this is a partition of  $[n]$  into two parts of size  $\Theta(n)$ . We also know for the root that  $\mathbf{SP}_A(\mathbf{Stab}(r)) = \{[n]\}$  because the circuit is stabilised by every permutation in  $\mathbf{Sym}_n$ . The bounds on the orbit-wise fan-in and fan-out degree that we assume in Theorem 2 allow us to prove that when we pass from any  $h_i$  to  $h_{i+1}$ , then  $\mathbf{SP}_A(\mathbf{Stab}(h_i)) \approx \mathbf{SP}_A(\mathbf{Stab}(h_{i+1}))$ , i.e. the supporting partitions hardly change (because if they did change more, then one would find that there must be more parent/child gates per orbit than allowed). Since  $\mathbf{SP}_A(\mathbf{Stab}(g))$  differs quite substantially from  $\mathbf{SP}_A(\mathbf{Stab}(r))$  but  $\mathbf{SP}_A(\mathbf{Stab}(h_i)) \approx \mathbf{SP}_A(\mathbf{Stab}(h_{i+1}))$  for every  $i$ , we can infer that all “intermediate partitions” between  $\mathbf{SP}_A(\mathbf{Stab}(g))$  and  $\mathbf{SP}_A(\mathbf{Stab}(r))$  appear for the gate stabilisers along the path. Then we show that one of these “intermediate partitions” which must appear as  $\mathbf{SP}_A(\mathbf{Stab}(h_i))$  for some  $i$  has properties which entail what we wanted, namely that  $\mathbf{Orbit}_{(h_i)}(h_{i+1})$  has even size. Thus, the edge  $(h_{i+1}, h_i)$  on the path satisfies what we were looking for. So in short, the proof works because: Firstly, in order to show that a path has an even number of automorphic images, it suffices to find one gate on the path where the number of symmetric predecessors is even. Secondly, whether or not this happens can be inferred from the supporting partition of the gate. Thirdly, the fan-in and fan-out bounds enable us to track very precisely how the supporting partitions along the path look like. With these arguments, we can show for a sufficient number of input gates  $g$ , that they do not contribute to the result of the XOR-computation because they have an even number of paths to  $r$  – thus, circuit property 3 in Theorem 2 is not satisfied when the other conditions are. Carrying out this proof sketch requires quite some work that we have swept under the carpet here; for example, one needs certain group-theoretic arguments similar to the proof of Theorem 5.2 B in [13] in order to be able to describe the stabiliser groups appropriately with alternating supporting partitions.

## 7 Conclusion and future research

We have shown that the definability of the CFI-query on a class  $\mathcal{K}$  of base graphs by means of a CFI-symmetric algorithm presupposes the existence of symmetric XOR-circuits with the properties from Theorem 1. We come close to proving the non-existence of these circuits for  $\mathcal{K}$  being the family of  $n$ -dimensional hypercubes. It remains as a problem for future research to improve this lower bound and close the gap between the circuit properties in Theorems 1 and 2 in order to separate the class of CFI-symmetric algorithms from PTIME. Once that is achieved, the next step would be to lift this circuit approach to *all* CPT-algorithms for the CFI-query, not just the CFI-symmetric ones. This is a potential route to eventually solve the extremely difficult problem of separating CPT from PTIME. In the full version of the paper, we also provide a generalised circuit construction which works for a larger class of h.f. sets than CFI-symmetric ones. These are sets whose  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ -symmetries and  $\mathbf{Aut}(G)$ -symmetries

go well together in the sense that the Boolean vector spaces describing the  $\mathbf{Aut}_{\text{CFI}}(\mathfrak{G})$ -stabilisers admit  $\mathbf{Aut}(G)$ -symmetric bases. However, one can prove that, unfortunately, the class of h.f. sets with this property is still not the full class of all CPT-definable sets.

A different approach towards CPT lower bounds could be to study the isomorphism problem of *multipedes* [25] instead of CFI-graphs. Such structures have been used to obtain lower bounds against individualization-refinement graph isomorphism algorithms. Multipedes differ from CFI-graphs in so far as they have no non-trivial automorphisms. Their inherent symmetries are rather given by counting-logic *types* (two tuples in a structure have the same *k-type* if they satisfy the same  $C^k$ -formulas). It could be that the circuit construction is adaptable to this kind of symmetry. This would be of particular interest in case that the (unordered) CFI-query turns out to be in fact CPT-definable; then, multipedes might still provide an example to separate CPT from PTIME.

---

### References

- 1 Matthew Anderson and Anuj Dawar. On symmetric circuits and fixed-point logics. *Theory of Computing Systems*, 60(3):521–551, 2017.
- 2 Albert Atserias, Andrei Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.
- 3 Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3):141–187, 1999.
- 4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.
- 5 Ashok Chandra and David Harel. Structure and complexity of relational queries. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 333–347. IEEE, 1980. doi:10.1109/SFCS.1980.41.
- 6 Anuj Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.
- 7 Anuj Dawar, Erich Grädel, and Wied Pakusa. Approximations of Isomorphism and Logics with Linear-Algebraic Operators. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 112:1–112:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.112.
- 8 Anuj Dawar, Erich Grädel, and Moritz Lichter. Limitations of the invertible-map equivalences. *Journal of Logic and Computation*, September 2022. doi:10.1093/logcom/exac058.
- 9 Anuj Dawar, David Richerby, and Benjamin Rossman. Choiceless Polynomial Time, Counting and the Cai–Fürer–Immerman graphs. *Annals of Pure and Applied Logic*, 152(1-3):31–50, 2008.
- 10 Anuj Dawar and Gregory Wilsenach. Symmetric Arithmetic Circuits. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.36.
- 11 Anuj Dawar and Gregory Wilsenach. Symmetric circuits for rank logic. *ACM Transactions on Computational Logic (TOCL)*, 23(1):1–35, 2021.
- 12 Anuj Dawar and Gregory Wilsenach. Lower Bounds for Symmetric Circuits for the Determinant. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:22, 2022. doi:10.4230/LIPIcs.ITCS.2022.52.
- 13 John Dixon and Brian Mortimer. *Permutation Groups*. Springer, New York, 1996.
- 14 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.
- 15 F. Gire and H.K. Hoang. An extension of fixpoint logic with a symmetry-based choice construct. *Information and Computation*, 144(1):40–65, 1998. doi:10.1006/inco.1998.2712.

- 16 E. Grädel, W. Pakusa, S. Schalthöfer, and L. Kaiser. Characterising Choiceless Polynomial Time with First-Order Interpretations. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 677–688, 2015.
- 17 Erich Grädel and Martin Grohe. Is Polynomial Time Choiceless? In *Fields of Logic and Computation II*, pages 193–209. Springer, 2015.
- 18 Martin Grohe. The quest for a logic capturing PTIME. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 267–271. IEEE, 2008. doi:10.1109/LICS.2008.11.
- 19 Martin Grohe, Pascal Schweitzer, and Daniel Wiebking. Deep Weisfeiler Leman, 2020. arXiv:2003.10935.
- 20 Yuri Gurevich. Logic and the Challenge of Computer Science. In *Current Trends in Theoretical Computer Science*. Computer Science Press, 1988.
- 21 William He and Benjamin Rossman. Symmetric formulas for products of permutations, 2022. arXiv:2211.15520.
- 22 Moritz Lichter. Separating Rank Logic from Polynomial Time. *J. ACM*, November 2022. doi:10.1145/3572918.
- 23 Moritz Lichter. Witnessed Symmetric Choice and Interpretations in Fixed-Point Logic with Counting, 2022. arXiv:2210.07869.
- 24 Moritz Lichter and Pascal Schweitzer. Choiceless Polynomial Time with Witnessed Symmetric Choice. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, LICS '22. Association for Computing Machinery, 2022. doi:10.1145/3531130.3533348.
- 25 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 138–150, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188900.
- 26 Benedikt Pago. Choiceless Computation and Symmetry: Limitations of Definability. In *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, 2021. doi:10.4230/LIPIcs.CSL.2021.33.
- 27 Benedikt Pago. Lower bounds for Choiceless Polynomial Time via Symmetric XOR-circuits, 2023. arXiv:2302.05426.
- 28 Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen, 2015.
- 29 Wied Pakusa, Svenja Schalthöfer, and Erkal Selman. Definability of Cai-Fürer-Immerman problems in Choiceless Polynomial Time. *ACM Transactions on Computational Logic (TOCL)*, 19(2):1–27, 2018. doi:10.1145/3154456.
- 30 Benjamin Rossman. Choiceless Computation and Symmetry. In *Fields of Logic and Computation*, pages 565–580. Springer, 2010.
- 31 Benjamin Rossman. Subspace-Invariant  $AC^0$  Formulas. *Logical Methods in Computer Science*, 15, 2019.
- 32 Svenja Schalthöfer. *Choiceless Computation and Logic*. PhD thesis, RWTH Aachen, 2020.



# A Super-Polynomial Separation Between Resolution and Cut-Free Sequent Calculus

Theodoros Papamakarios ✉

Department of Computer Science, University of Chicago, IL, USA

---

## Abstract

We show a quadratic separation between resolution and cut-free sequent calculus width. We use this gap to get, for the first time, first, a super-polynomial separation between resolution and cut-free sequent calculus for refuting CNF formulas, and secondly, a quadratic separation between resolution width and monomial space in polynomial calculus with resolution. Our super-polynomial separation between resolution and cut-free sequent calculus only applies when clauses are seen as disjunctions of unbounded arity; our examples have linear size cut-free sequent calculus proofs writing, in a particular way, their clauses using binary disjunctions. Interestingly, this shows that the complexity of sequent calculus depends on how disjunctions are represented.

**2012 ACM Subject Classification** Theory of computation → Proof complexity

**Keywords and phrases** Proof Complexity, Resolution, Cut-free LK

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.74

**Related Version** *Full Version*: <https://ecc.weizmann.ac.il/report/2021/176/>

**Acknowledgements** We wish to thank Alexander Razborov for numerous suggestions and remarks that greatly improved the presentation of the paper.

## 1 Introduction

Whether cut-free sequent calculus can polynomially simulate resolution for refuting CNF formulas is a question existing since the beginnings of proof complexity. It was first raised in [13] and iterated e.g. in [25]. Cook and Reckhow [13] show that in the tree-like case, there are examples where resolution can have exponentially smaller proofs. Arai, Pitassi and Urquhart [3] point out that the answer may heavily depend on how clauses are represented. A clause consisting of the literals, say  $\ell_1, \ell_2, \ell_3, \ell_4$ , can be seen as either a single disjunction of arity four, or as a series of applications of binary disjunctions, for example  $(\ell_1 \vee \ell_2) \vee (\ell_3 \vee \ell_4)$ , and this can have a profound impact on the complexity of sequent calculus proofs. The result of Cook and Reckhow above applies in the case where clauses are seen as single disjunctions of unbounded arity, or the case where the order in which the binary disjunctions are applied is fixed. If we are free to choose the order, then tree-like cut-free sequent calculus can quasi-polynomially simulate tree-like resolution, and this is optimal [3]. In the DAG-like case, and if we are free to choose the order in which binary disjunctions are applied, Reckhow [21] shows that cut-free sequent calculus can polynomially simulate regular resolution, and Arai [2] shows that it can polynomially simulate resolution for refuting  $k$ -CNF formulas  $F$ , where  $k$  grows at most logarithmically as a function of the size of the shortest resolution refutation of  $F$ . However, the general question has remained unresolved.

We define the width of a sequent calculus proof as the maximum number of formulas occurring in a sequent of the proof. This definition extends in a natural way the concept of the width of a resolution proof to stronger proof systems. Furthermore, it allows for a simple, abstract characterization of sequent calculus width generalizing the characterization of Atserias and Dalmau for resolution width [4]. Using this characterization, we show a quadratic gap between resolution width and cut-free sequent calculus width. Resolution is a



© Theodoros Papamakarios;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 74; pp. 74:1–74:15

Leibniz International Proceedings in Informatics



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

sequent calculus system that has only atomic cuts, so this says that including atomic cuts in cut-free sequent calculus can shorten the width of proofs. Utilizing then this gap, we show a super-polynomial separation between cut-free sequent calculus and resolution size. To put it in other words, atomic cuts can super-polynomially decrease the size of proofs. This result applies only when clauses are seen as disjunctions of unbounded arity. There is a way to write the clauses in our examples using binary disjunctions, so that the resulting formulas have linear size cut-free sequent calculus refutations. Thus, as it was already known for the tree-like case, the complexity of sequent calculus proofs can depend on how disjunctions are represented.

Several notions of width have been used to show space lower bounds in different proof systems, demonstrating a close relationship between the two measures [5, 14, 4, 11, 12, 10, 9, 17, 20]. We note that our characterization of cut-free sequent calculus width for refuting CNF formulas coincides with the concept of dynamic satisfiability, introduced by Esteban, Galesi and Messner [14] as a tool for proving space lower bounds in resolution and  $k$ -DNF resolution. It is easily seen that dynamic satisfiability is a weakened version of resolution width. How much weaker however is a question that has not been addressed. We show that it is strictly weaker, the quadratic gap between resolution and cut-free sequent calculus width being a quadratic gap between the two. Furthermore, our basic construction extends to stronger versions of dynamic satisfiability used to prove monomial space lower bounds in algebraic proof systems [11, 12], allowing us to make progress towards separating resolution width from monomial space.

To put things into perspective, Atserias and Dalmau [4] show that for a  $k$ -CNF formula  $F$ ,  $W(F \vdash \perp)$ , the minimum width, and  $\text{CSpace}(F \vdash \perp)$ , the minimum clause space needed to refute  $F$  in resolution satisfy

$$W(F \vdash \perp) \leq \text{CSpace}(F \vdash \perp) + k, \quad (1.1)$$

and Galesi, Kołodziejczyk and Thapen [17] show a similar relation between resolution width and the minimum monomial space needed to refute  $F$  in polynomial calculus with resolution:

$$W(F \vdash \perp) \leq O\left((\text{MSpace}(F \vdash \perp))^2\right) + k. \quad (1.2)$$

Ben-Sasson and Nordström [6, 7] give for every  $n$ , a formula  $F$  of size  $n$  such that  $W(F \vdash \perp) = O(1)$  and  $\text{CSpace}(F \vdash \perp) = \Omega(n/\log n)$ , rendering a relation between resolution width and clause space in the opposite direction impossible. Whether clause space can be meaningfully bounded in terms of monomial space is unknown, but the two measures are related in a more indirect way: they coincide up to polynomial and  $\log n$  factors once regularized, meaning that a super-polynomial separation of them would imply a strong trade-off between monomial space and size [20]. Despite however this close relationship, no separation between width and monomial space is currently known – the techniques of [6, 7] in particular fail to generalize to the case of monomial space. Our result shows a quadratic separation between the two.

## 2 Sequent calculus

The sequent calculus was introduced by Gentzen to formulate and prove his famous cut-elimination theorem. Many authors describe it as the most elegant proof system, and indeed, it illustrates the symmetries of logic at the level of syntax, like no other system.

Sequent calculus's version for classical logic is often denoted by LK. We shall use LK to denote its propositional part. LK operates with *sequents*. A sequent is a tuple of the form  $(\Gamma, \Delta)$ , where  $\Gamma$  and  $\Delta$  are finite sets of formulas. Traditionally, a sequent  $(\Gamma, \Delta)$  is written as  $\Gamma \rightarrow \Delta$ . This is to remind us its semantic interpretation:  $\Gamma \rightarrow \Delta$  is to be read as “if all formulas in  $\Gamma$  are true, then at least one formula in  $\Delta$  is true”.

Let us present the rules of the system. In what follows,  $A, B$  represent arbitrary formulas, and  $\Gamma, \Delta, \Gamma', \Delta'$  represent finite sets of formulas. Sets are written in a quite plain manner: We write  $\Gamma, A$  instead of  $\Gamma \cup \{A\}$ ,  $A$  instead of  $\{A\}$ ,  $A, B$  instead of  $\{A, B\}$  and so on. The *axioms* of LK are all sequents of the form  $A \rightarrow A$ . Of the inference rules, first we have a rule, which allows us to add formulas to the left or right part of a sequent. This rule is called the *thinning* or *weakening* rule and has the form

$$\frac{\Gamma \rightarrow \Delta}{\Gamma' \rightarrow \Delta'}$$

i.e., from  $\Gamma \rightarrow \Delta$  derive  $\Gamma' \rightarrow \Delta'$ , where  $\Gamma \subseteq \Gamma'$  and  $\Delta \subseteq \Delta'$ . Next, we have rules for each connective. These come in pairs; a connective is treated differently according to which side of its sequent it appears. The rules for the connectives  $\wedge$ ,  $\vee$  and  $\neg$  are shown in Table 1.

■ **Table 1** The analytic LK rules.

$$\begin{array}{l} \neg\text{L} : \frac{\Gamma, \neg A \rightarrow \Delta, A}{\Gamma, \neg A \rightarrow \Delta} \qquad \neg\text{R} : \frac{\Gamma, A \rightarrow \Delta, \neg A}{\Gamma \rightarrow \Delta, \neg A} \\ \wedge\text{L}_1 : \frac{\Gamma, A \wedge B, A \rightarrow \Delta}{\Gamma, A \wedge B \rightarrow \Delta} \qquad \wedge\text{R} : \frac{\Gamma \rightarrow \Delta, A \wedge B, A \quad \Gamma \rightarrow \Delta, A \wedge B, B}{\Gamma \rightarrow \Delta, A \wedge B} \\ \wedge\text{L}_2 : \frac{\Gamma, A \wedge B, B \rightarrow \Delta}{\Gamma, A \wedge B \rightarrow \Delta} \\ \vee\text{R}_1 : \frac{\Gamma \rightarrow \Delta, A \vee B, A}{\Gamma \rightarrow \Delta, A \vee B} \qquad \vee\text{L} : \frac{\Gamma, A \vee B, A \rightarrow \Delta \quad \Gamma, A \vee B, B \rightarrow \Delta}{\Gamma, A \vee B \rightarrow \Delta} \\ \vee\text{R}_2 : \frac{\Gamma \rightarrow \Delta, A \vee B, B}{\Gamma \rightarrow \Delta, A \vee B} \end{array}$$

These rules are called *analytic*, and they already form a complete proof system for proving tautologies; we shall call this system *cut-free* LK or  $\text{LK}^-$ . Finally, there is the *cut rule*:

$$\frac{\Gamma, A \rightarrow \Delta \quad \Gamma \rightarrow \Delta, A}{\Gamma \rightarrow \Delta}. \tag{2.1}$$

So, already having a proof of, say  $\rightarrow B$ , we may use it to prove  $\rightarrow A$ :  $\rightarrow A$  can be derived from  $B \rightarrow A$  and  $\rightarrow B$  via the cut rule, and now to prove  $\rightarrow A$ , we need to prove the weaker formula  $B \rightarrow A$ .  $B$  can be anything. It doesn't need to have any intuitive relation to  $A$ , but even as such, it might be the case that a proof of  $B \rightarrow A$  is much shorter than a proof of  $\rightarrow A$ . Gentzen's cut elimination theorem says that there is always an effective procedure of eliminating all applications of the cut rule from a proof, making it purely analytic. We refer to the formula  $A$  in applications of rule (2.1) as the formula being cut, or as the *cut formula*.

An LK proof of a sequent  $\sigma$  is a derivation of  $\sigma$  starting with the axioms and applying the rules of LK. We imagine starting with the axioms at the bottom, and going to the top by applying the LK rules. More formally, an LK proof of  $\sigma$  is a sequence consisting of sequents

that ends with  $\sigma$ , in which every sequent is either an axiom, or results from previous sequents by one of the LK rules. An  $LK^-$  proof is an LK proof that never uses the cut rule. We may view proofs as DAGs, by drawing edges from premises to conclusions in applications of the inference rules. If the DAG corresponding to a proof is a tree, we shall refer to the proof as being *tree-like*.

**3** Sequent calculus as a satisfiability algorithm

It will be particularly convenient to consider the following view of LK. Following Smullyan [23], let us write a sequent  $A_1, \dots, A_k \rightarrow B_1, \dots, B_\ell$  as  $TA_1, \dots, TA_k, FB_1, \dots, FB_\ell$ . That is, we annotate the formulas appearing on the left side of a sequent by  $T$ , the formulas appearing on its right side by  $F$ , and conjoin the two sides to form a single set.  $T$  and  $F$  stand for true and false respectively –  $TA$  should be thought of as asserting that  $A$  is true and  $FA$  as asserting that  $A$  is false.

Annotated formulas that are not annotated variables are naturally divided into two groups: those of a conjunctive and those of a disjunctive type. Formulas of the form  $TA \wedge B$ ,  $FA \vee B$ ,  $T\neg A$  or  $F\neg A$  belong to the former group, and those of the form  $TA \vee B$  or  $FA \wedge B$  to the latter. We use the letter “ $\alpha$ ” to stand for an arbitrary annotated formula of conjunctive type, and the letter “ $\beta$ ” to stand for an arbitrary annotated formula of disjunctive type. We define the components  $\alpha_i$  of a formula  $\alpha$  and the components  $\beta_i$  of a formula  $\beta$  as shown in Table 2.

■ **Table 2** Smullyan’s notation.

$\alpha$	$\alpha_1$	$\alpha_2$	$\beta$	$\beta_1$	$\beta_2$
$TA \wedge B$	$TA$	$TB$	$FA \wedge B$	$FA$	$FB$
$FA \vee B$	$FA$	$FB$	$TA \vee B$	$TA$	$TB$
$T\neg A$	$FA$				
$F\neg A$	$TA$				

These provisions allow on the one hand for an extremely concise description of the rules of Table 1; they can be written as:

$$\frac{S, \alpha, \alpha_1}{S, \alpha}, \quad \frac{S, \alpha, \alpha_2}{S, \alpha}, \quad \frac{S, \beta, \beta_1 \quad S, \beta, \beta_2}{S, \beta}.$$

More importantly, they reveal an algorithmic interpretation of LK. An LK proof, seen from the top to the bottom, i.e. from the sequent  $\sigma := A_1, \dots, A_k \rightarrow B_1, \dots, B_\ell$  it is proving to the axioms, describes the execution of an algorithm that tries to find a truth assignment (or more generally a model) that falsifies  $\sigma$ . The algorithm begins with  $\sigma$  written as  $TA_1, \dots, TA_k, FB_1, \dots, FB_\ell$ , asserting that there is an assignment that makes all  $A_i$  true and all  $B_i$  false, or equivalently, an assignment that falsifies  $\sigma$ . Then it keeps expanding this set, by applying the LK rules in reverse, that is from the conclusion to the premises. This expansion takes the form of a tree (or a DAG if we identify nodes labelled by the same set). At any point, we may choose a leaf labelled by  $S, \alpha$  and add to it a single child labelled by  $S, \alpha, \alpha_i$ , as any assignment satisfying  $\alpha$  must also satisfy every  $\alpha_i$ . Or we may choose a leaf labelled by  $S, \beta$  and add to it two children, one labelled by  $S, \beta, \beta_1$  and the other by  $S, \beta, \beta_2$ , as any assignment satisfying  $\beta$  must either satisfy  $\beta_1$  or  $\beta_2$ . The thinning rule allows the algorithm to forget information: We may add to a leaf labelled by  $S$ , a child labelled by a subset of  $S$ . Finally, the cut rule allows us to add to a leaf labelled by  $S$ , two children,

one labelled by  $S, T A$  and the other by  $S, F A$  for any formula  $A$ , as every assignment must satisfy either  $A$  or  $\neg A$ . This may greatly facilitate the search procedure. If at any point a set of the form  $T A, F A$  is reached, then the search process may terminate at that particular branch, as no assignment can set  $A$  to both true and false. Notice that the contradiction  $T A, F A$  corresponds to the axiom  $A \rightarrow A$ . A tree (or DAG) constructed this way, every branch of which ends with a leaf labelled by a set of the form  $T A, F A$ , is an LK proof of  $\sigma$ .

A depth-first implementation of the algorithm described above is shown as Algorithm 1 below. Algorithm 1 is called on a sequent represented as a set of annotated formulas. It

■ **Algorithm 1** The LK algorithm.

---

```

procedure LK( $S$ )
  if  $S$  contains both  $T A$  and  $F A$  for some formula  $A$  then
    return false
  if for every  $\alpha \in S$ , all  $\alpha_i \in S$  and for every  $\beta \in S$ , there is a  $\beta_i \in S$  then
    return true
  go to either 1, 2 or 3
  1. select an  $S' \subseteq S$  and return LK( $S'$ )
  2. select an arbitrary formula  $A$  and return LK( $S, T A$ ) or LK( $S, F A$ )
  3. select an  $A \in S$ 
  if  $A = \alpha$  then
    select a component  $\alpha_i$  and return LK( $S, \alpha_i$ )
  if  $A = \beta$  then
    return LK( $S, \beta_1$ ) or LK( $S, \beta_k$ )

```

---

chooses at each recursive call non-deterministically what rule to apply and which formula to apply it to. If false is returned then there is an LK proof of our initial sequent; if no such proof exists, then there is an assignment that falsifies our initial sequent and Algorithm 1 is able to find it, returning true. As presented, line 1, corresponding to the thinning rule, is redundant. However, incorporating memoization, that is the ability to stop the search when a set  $S$  has already been encountered in a previous recursive call that has returned, effectively identifying nodes labelled by the same set, this line makes it possible to greatly prune the search for a falsifying assignment. In terms of proofs, DAG-like proofs may be shorter than tree-like proofs. The key point in analyzing the correctness of Algorithm 1 (or equivalently the completeness of LK), is that when at the base case true is returned, we can create an assignment consistent with  $S$  by setting for each formula  $A$ ,  $A$  to true if  $T A \in S$ , and false otherwise.

#### 4 The width of sequent calculus proofs

We define the *width* of a sequent as the number of formulas it contains, and the width of a sequent calculus proof as the maximum of the widths of the sequents it contains. It is not hard to see that for any provable sequent  $S_0$ , there is an LK proof of  $S_0$  of width a constant plus the width of  $S_0$ . The concept of the minimum width needed to prove a sequent becomes non-trivial only if we restrict the class of cut formulas we are allowed to use. We shall be mainly interested in the minimum width over all  $LK^-$  proofs of  $S_0$ , which we denote by  $W_{LK^-}(\vdash S_0)$ .

We are going to give a characterization of  $W_{LK^-}(\vdash S_0)$  in terms of the definition below. In what follows, sequents are viewed as in the above section, viz. as sets of annotated formulas.

► **Definition 4.1.** Following the terminology of [23], let us call a sequent  $S_0$  *analytically  $k$ -consistent* if there is a set of sequents  $\mathcal{S}$  containing  $S_0$  and such that for each  $S \in \mathcal{S}$ :

1. for any formula  $A$ ,  $S$  does not contain both  $T A$  and  $F A$ ;
2.  $S' \subseteq S \implies S' \in \mathcal{S}$ ;
3.  $|S| < k$  &  $\alpha \in S \implies S, \alpha_i \in \mathcal{S}$  for every component  $\alpha_i$  of  $\alpha$ ;
4.  $|S| < k$  &  $\beta \in S \implies S, \beta_i \in \mathcal{S}$  for some component  $\beta_i$  of  $\beta$ .

If the following condition is also satisfied, then we call  $S_0$  *synthetically  $k$ -consistent* with respect to the set  $\mathcal{C}$ :

5.  $|S| < k \implies S, T A \in \mathcal{S}$  or  $S, F A \in \mathcal{S}$  for any formula  $A \in \mathcal{C}$ .

It is often helpful to see definitions such as the above, as describing a strategy for the adversary, in a game between a prover and an adversary played on a formula/sequent/set of formulas. In this case, the game is as follows: The configurations of the game are sequents. The initial configuration is  $S_0$ . In every round, the prover either deletes some formulas in the current sequent  $S$ , or selects an  $\alpha$ -formula in  $S$  and adds a component of it to  $S$ , or selects a  $\beta$  formula, in which case the adversary adds a component of it to  $S$ . Allowing condition 5, the prover may choose an arbitrary formula  $A \in \mathcal{C}$  and the adversary must respond by adding either  $T A$  or  $F A$  to  $S$ . The game ends with prover winning once  $S$  contains  $T A$  and  $F A$  for some formula  $A$ . The prover can always win provided that  $S_0$  is provable. The question is: given a bound  $k$ , can she win always maintaining that the width of  $S$  is at most  $k$ ? Definition 4.1 describes a strategy for the adversary, permitting the prover from winning when she maintains that bound.

► **Theorem 4.1.** *Suppose that  $|S_0| \leq k$ . Then  $S_0$  is analytically  $k$ -consistent if and only if  $W_{\text{LK}^-}(\vdash S_0) > k$ . It is synthetically  $k$ -consistent with respect to  $\mathcal{C}$  if and only if every LK proof of  $S_0$  in which every cut formula belongs to  $\mathcal{C}$  has width more than  $k$ .*

**Proof.** Let us only show the former sentence. Suppose first that  $S_0$  is analytically  $k$ -consistent, and let  $\mathcal{S}$  be the set of sequents witnessing this. We will show that in every tree-like LK $^-$  derivation (not necessarily beginning with axioms)  $\tau$  of  $S_0$ , of width at most  $k$ , there is an initial sequent (i.e. one appearing as a leaf) in  $\mathcal{S}$ . From the first condition of Definition 4.1 that sequent is not an axiom, thus  $\tau$  is not a proof. It is enough to show this for tree-like derivations, since a DAG-like derivation can be transformed into a tree-like one without increasing the width.

**Base case.** If  $\tau$  contains just  $S_0$ , then we are done since  $S_0 \in \mathcal{S}$ .

**Inductive step.** Take some initial sequents  $S_1, \dots, S_r$  from which a sequent  $S$  is derived via an inference rule  $\rho$ , and remove them to get the derivation  $\tau'$ . From the induction hypothesis, there is an initial sequent in  $\tau'$  that belongs to  $\mathcal{S}$ . If that sequent is not  $S$ , then it also appears in  $\tau$  and we are done. Otherwise, we have the following cases according to what rule  $\rho$  is:

**Case 1.** If it is the weakening rule, and thus  $r = 1$  and  $S_1 \subseteq S$ , then from the second condition of Definition 4.1,  $S_1 \in \mathcal{S}$ .

**Case 2.** If  $\rho$  is the  $\alpha$ -rule, and thus  $r = 1$ ,  $\alpha \in S$  and  $S_1 = S, \alpha_i$  for some  $\alpha_i$ , then since  $\tau$  has width at most  $k$ ,  $|S| < k$ , and hence from the third condition of Definition 4.1,  $S_1 \in \mathcal{S}$ .

**Case 3.** If  $\rho$  is the  $\beta$ -rule, and thus  $\beta \in S$  and each  $S_i$  is of the form  $S, \beta_i$ , then again  $|S| < k$ , and from the fourth condition of Definition 4.1, some  $S_i$  belongs to  $\mathcal{S}$ .

Now suppose that  $W_{\text{LK}^-}(\vdash S_0) > k$ . Set

$$\mathcal{S} := \{S \mid |S| \leq k \text{ \& } W_{\text{LK}^-}(\vdash S) > k\}.$$

Clearly  $S_0 \in \mathcal{S}$ . We will show that  $\mathcal{S}$  satisfies the conditions 1–4 of Definition 4.1. For each  $S \in \mathcal{S}$ , first  $S$  cannot contain  $T A$  and  $F A$  for some  $A$ . This is so, because such a sequent is a weakening of an axiom, and having size at most  $k$ , it has a proof of width at most  $k$ . For the closure under subsets, if  $S' \subseteq S$ , then  $W_{\text{LK}^-}(\vdash S') > k$ , for otherwise  $W_{\text{LK}^-}(\vdash S) \leq k$  since  $S$  follows from  $S'$  via the weakening rule. For the  $\alpha$  condition, if  $\alpha \in S$  and  $|S| < k$ , then for each  $\alpha_i$  it must be that  $S, \alpha_i \in \mathcal{S}$ , for otherwise  $W_{\text{LK}^-}(\vdash S) \leq k$  since  $S$  follows from  $S, \alpha_i$  via the  $\alpha$ -rule. Finally, if  $\beta \in S$  and  $|S| < k$ , then there must be a  $\beta_i$  such that  $S, \beta_i \in \mathcal{S}$ , otherwise  $W_{\text{LK}^-}(\vdash S) \leq k$ , since  $S$  follows from all  $S, \beta_i$  via the  $\beta$ -rule. ◀

## 5 LK<sup>-</sup> for refuting CNF formulas and resolution

A *literal* is a propositional variable  $x$ , or the negation of propositional variable  $\neg x$ . We let  $\bar{x} \stackrel{\text{def}}{=} \neg x$  and  $\overline{\neg x} \stackrel{\text{def}}{=} x$ . A *clause* is a disjunction (possibly empty) of literals, and a *CNF formula* is a conjunction of clauses. The ordering of literals in a clause does not matter, so that the clause  $x \vee y$  is considered to be the same as  $y \vee x$ . The *width*,  $W(F)$ , of a CNF formula  $F$  is the number of literals in the largest clause of  $F$ . A CNF formula of width at most  $k$  is called a  $k$ -CNF formula.

Refuting a CNF formula  $F = C_1 \wedge \dots \wedge C_m$  means proving that the clauses  $C_i$  cannot be simultaneously satisfied, that is, it means proving  $C_1, \dots, C_m \rightarrow$ . LK<sup>-</sup> for proving such sequents has the following form. Of the rules in Table 1, the only one that is relevant is  $\vee\text{L}$ , which now, seeing clauses as disjunctions of unbounded arity, has as many premises as the number of literals in the clause it is deriving. Moreover, there is no reason to always carry the clauses  $C_i$  in sequents. We may as well delete them from every sequent, but keep in our mind that they are implicitly there. What remains are sequents of the form  $\ell_1, \dots, \ell_r \rightarrow$ , where the  $\ell_i$ 's are literals, and such sequents are nothing other than clauses. To be explicit, the axioms of the resulting system are clauses of the form  $x \vee \neg x$ , and the inference rules are the weakening rule, from  $C$  infer  $C \vee D$ , and

$$\frac{C \vee \bar{\ell}_1 \quad \dots \quad C \vee \bar{\ell}_r}{C}, \quad (5.1)$$

where  $C$  and  $D$  are clauses and  $\ell_1 \vee \dots \vee \ell_r$  is a clause of the formula we are refuting. A proof of  $C_1, \dots, C_m \rightarrow$ , in other words a *refutation* of  $F = C_1 \wedge \dots \wedge C_m$ , in LK<sup>-</sup>, is a derivation of the empty clause using the above rules. The *size* of such a derivation is the number of clauses it contains, and its *width* is the size of the largest clause occurring in it. We shall denote by  $S_{\text{LK}^-}(F \vdash \perp)$  and  $W_{\text{LK}^-}(F \vdash \perp)$  and the minimum size and the minimum width respectively over all LK<sup>-</sup> refutations of  $F$ .

*Resolution* is the system we get by adding to the above system the cut rule (2.1), where the cut formula  $A$  is restricted to be a propositional variable:

$$\frac{C \vee x \quad C \vee \neg x}{C}. \quad (5.2)$$

We may make a proof in this system “cut-only”, by pushing all applications of the rule (5.1) at the bottom levels. Namely, we can simulate rule (5.1) by (5.2) as follows: Start with  $\ell_1 \vee \dots \vee \ell_r$ , derive from it and  $C \vee \bar{\ell}_1$ ,  $C \vee \ell_2 \vee \dots \vee \ell_r$ , then derive from  $C \vee \ell_2 \vee \dots \vee \ell_r$  and  $C \vee \bar{\ell}_2$ ,  $C \vee \ell_3 \vee \dots \vee \ell_r$ , and so on, until  $C$  is derived. Now the leaves containing clauses of  $F$  and these can be derived from axioms by (5.1). Deleting all axioms, and incorporating the weakening rule into (5.2), writing it as

$$\frac{C \vee x \quad D \vee \neg x}{C \vee D}, \quad (5.3)$$

we get the usual presentation of resolution, where, instead of deriving  $F \rightarrow$ , the empty clause is derived taking the clauses of  $F$  as axioms: A *resolution refutation* of a CNF formula  $F$  is a derivation of the empty clause from the clauses of  $F$ , using only the rule (5.3). We shall denote by  $W_R(F \vdash \perp)$  and  $S_R(F \vdash \perp)$  the minimum width and minimum size respectively, over all resolution refutations of  $F$ , and by  $S_{TR}(F \vdash \perp)$  the minimum size, over all tree-like resolution refutations of  $F$ .

## 6 Dynamic satisfiability

Adapting Definition 4.1 for resolution we get the characterization of [4] for resolution width. Adapting it for  $LK^-$  restricted to refuting CNF formulas, we get the definition of dynamic satisfiability from [14]. Namely, let us call sets of literals that do not contain contradictory literals *partial assignments*. We think of the assignment, say  $\{x, \neg y, z\}$ , as making  $x$  true,  $y$  false and  $z$  true. A partial assignment satisfies a clause  $C$ , if it contains a literal of  $C$ . It falsifies  $C$  if it contains  $\bar{\ell}$  for every  $\ell$  in  $C$ . We get:

► **Definition 6.1** [14]. Let  $F$  be a CNF formula, and let  $k$  be a natural number.  $F$  is said to be *k-dynamically satisfiable* if there is a non-empty set  $\mathcal{A}$  of partial assignments to its variables such that for every assignment  $\alpha \in \mathcal{A}$ ,

1. if  $\alpha' \subseteq \alpha$  then  $\alpha' \in \mathcal{A}$ ;
2. if  $|\alpha| < k$  and  $C$  is a clause of  $F$ , then there is an  $\alpha' \supseteq \alpha$  in  $\mathcal{A}$  that satisfies  $C$ .

Theorem 4.1 in particular, becomes:

► **Theorem 6.1.** *A CNF  $F$  is k-dynamically satisfiable if and only if  $W_{LK^-}(F \vdash \perp) > k$ .*

In the game corresponding to Definition 6.1, prover chooses in each round a clause of  $F$ , and the adversary responds by choosing a literal in that clause, which adds to the current assignment. Again, the closure under subsets condition corresponds to the ability of the prover to delete at any round literals from the current assignment. The prover wins once the current assignment falsifies a clause of  $F$ .

We get a characterization of resolution width by having the prover selecting variables instead of clauses, and the adversary responding by giving values to them. More specifically, in every round the prover selects a variable  $x$  of  $F$ . Then the adversary selects either  $x$  or  $\neg x$ , and the prover updates the current assignment  $\alpha$  by deleting (if she wants) literals and adding the choice of the adversary. Again the prover wins once  $\alpha$  falsifies a clause of  $F$ . She can win always maintaining  $|\alpha| < k$  if and only if  $W_R(F \vdash \perp) \leq k$  [4].

Notice that, if  $W(F)$  is small, the prover in the second game is more powerful. Namely, we have  $W_R(F \vdash \perp) \leq W_{LK^-}(F \vdash \perp) + W(F) - 1$ . We already saw this when we explained how the resolution rule can simulate (5.1). In terms of games, the argument goes as follows: When the prover in the first game selects a clause  $C$ , the prover in the second game can start selecting, one by one the variables of  $C$ . If the game does not end, then the current assignment satisfies  $C$ , and then the prover can delete literals to match the assignments in the two games.

Definition 6.1 was introduced in [14] as a tool for proving space lower bounds in resolution and  $k$ -DNF resolution. The following definition is from [15, 1]. A *memory configuration* in resolution, is a set of clauses. A resolution refutation of a CNF formula  $F$ , in configurational

form, is a sequence  $\mathcal{M}_1, \dots, \mathcal{M}_t$  of configurations where  $\mathcal{M}_1$  is empty,  $\mathcal{M}_t$  contains the empty clause and for  $i > 1$ ,  $\mathcal{M}_i$  is obtained from  $\mathcal{M}_{i-1}$  by one of the following rules:

**Axiom download:**  $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ , where  $C$  is a clause of  $F$ .

**Inference:**  $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ , where  $C$  is derived from clauses in  $\mathcal{M}_{i-1}$  by the resolution rule.

**Erasure:**  $\mathcal{M}_i \subseteq \mathcal{M}_{i-1}$ .

The clause space of such a refutation is  $\max_{1 \leq i \leq t} |\mathcal{M}_i|$ . The clause space of a CNF formula  $F$ , denoted by  $\text{CSpace}(F \vdash \perp)$ , is the minimum clause space, over all refutations, in configurational form, of  $F$ .

► **Theorem 6.2** [14]. *If  $F$  is  $k$ -dynamically satisfiable, then  $\text{CSpace}(F \vdash \perp) \geq k$ .*

We thus have

$$W_R(F \vdash \perp) - W(F) + 1 \leq W_{\text{LK}^-}(F \vdash \perp) \leq \text{CSpace}(F \vdash \perp). \quad (6.1)$$

It is shown in [6] that there are 6-CNF formulas  $F$  of size  $O(n)$  such that  $W_R(F \vdash \perp) = O(1)$  and  $\text{CSpace}(F \vdash \perp) = \Omega(n/\log n)$ . It is easy to show that  $W_{\text{LK}^-}(F \vdash \perp) = O(1)$ , thus these formulas in fact provide a gap between  $W_{\text{LK}^-}(F \vdash \perp)$  and  $\text{CSpace}(F \vdash \perp)$ . The question of whether there is a gap between  $W_R(F \vdash \perp)$  and  $W_{\text{LK}^-}(F \vdash \perp)$  has not been addressed, and it is what we will deal with next.

## 7 A quadratic gap between $\text{LK}^-$ and resolution width

Let  $F = \bigwedge_{i=1}^s C_i$  and  $G = \bigwedge_{i=1}^t D_i$  be unsatisfiable CNF formulas. We define

$$F \times G \stackrel{\text{def}}{=} \bigwedge_{i=1}^s \bigwedge_{j=1}^t (C_i \vee D_j).$$

$F \times G$  is the CNF expansion of the formula  $F \vee G$ , which is also unsatisfiable.

Remarkably,  $\text{LK}^-$  width and resolution width exhibit a different behavior with respect to this construction. This disparity ultimately relies on the fact that the cut rule gives us the ability to combine given proofs into a more complicated proof.

On one hand, we have:

► **Lemma 7.1.** *If  $F$  and  $G$  are over disjoint sets of variables, then*

$$W_{\text{LK}^-}(F \times G \vdash \perp) \geq W_{\text{LK}^-}(F \vdash \perp) + W_{\text{LK}^-}(G \vdash \perp) - 1.$$

**Proof.** Suppose that  $F$  is  $k$ -dynamically satisfiable,  $G$  is  $\ell$ -dynamically satisfiable, and let  $\mathcal{A}$  and  $\mathcal{B}$  respectively be sets witnessing this. We need to show that  $F \times G$  is  $(k+\ell)$ -dynamically satisfiable, that is we need to find a set satisfying the conditions of Definition 6.1 for the parameter  $k+\ell$ . We claim that

$$\mathcal{C} := \{\alpha \cup \beta \mid \alpha \in \mathcal{A} \ \& \ \beta \in \mathcal{B}\}$$

is such a set. Closure under subsets immediately follows from the fact that  $\mathcal{A}$  and  $\mathcal{B}$  are closed under subsets. For the second condition, suppose that  $\gamma \in \mathcal{C}$ ,  $|\gamma| < k+\ell$ , and let  $C_i \vee D_j$  be a clause of  $F \times G$ , where  $C_i$  is a clause of  $F$  and  $D_j$  is a clause of  $G$ . Since  $\gamma \in \mathcal{C}$ , there is an  $\alpha \in \mathcal{A}$  and a  $\beta \in \mathcal{B}$  such that  $\gamma = \alpha \cup \beta$ . Moreover, since  $|\gamma| < k$  and  $F$  and  $G$  do not share variables, either  $|\alpha| < k$  or  $|\beta| < \ell$ . In the first case there is an  $\alpha' \supseteq \alpha$  in  $\mathcal{A}$  satisfying  $C_i$ , and thus  $\alpha' \cup \beta$  is an assignment in  $\mathcal{C}$  satisfying  $C_i \vee D_j$ . In the second case there is a  $\beta' \supseteq \beta$  in  $\mathcal{B}$  satisfying  $D_j$ , and thus  $\alpha \cup \beta'$  is an assignment in  $\mathcal{C}$  satisfying  $C_i \vee D_j$ . ◀

For resolution on the other hand, we have:

► **Lemma 7.2.**  $W_R(F \times G \vdash \perp) \leq \max\{W_R(F \vdash \perp) + W(G), W_R(G \vdash \perp)\}$ .

**Proof.** Let  $\pi$  and  $\rho$  be resolution refutations of  $F$  and  $G$  respectively, both of minimum width. Replacing every clause  $C$  in  $\pi$  with  $C \vee D_i$  we get a resolution proof  $\pi_i$  of  $D_i$  from  $F \times G$ .  $\pi_i$  has width at most  $W_R(F \vdash \perp) + W(F)$ . Replacing then every clause  $D_i$  in  $\rho$  with  $\pi_i$  we get a resolution refutation of  $F \times G$  with the stated width. ◀

Choosing an appropriate seed and iterating, we get our result.

► **Theorem 7.1.** *There are CNF formulas  $G$  with  $n^2$  variables, size  $O(n)^n$ , and such that  $W_R(G \vdash \perp) = O(n)$  and  $W_{LK^-}(G \vdash \perp) = \Omega(n^2)$ .*

**Proof.** Let  $F$  be a CNF formula with  $n$  variables, width  $O(1)$ , size  $\Theta(n)$ , and such that  $W_R(F \vdash \perp) = \Theta(n)$ . Such formulas exist from e.g. [8]. Consider the formula  $F^n := F_1 \times \cdots \times F_n$ , where the  $F_i$ 's are copies of  $F$  over mutually disjoint sets of variables. From Lemma 7.2,  $W_R(F^n \vdash \perp) = O(n)$ . On the other hand  $W_{LK^-}(F \vdash \perp) = \Omega(n)$  from (6.1), and hence from Lemma 7.1,  $W_{LK^-}(F^n \vdash \perp) = \Omega(n^2)$ . ◀

## 8 Separating resolution width from monomial space

Monomial space is a generalized version of clause space. While configurations in the case of clause space are sets of clauses, for monomial space, arbitrary linear combinations, over a field  $\mathbb{F}$ , of clauses are allowed as the contents of a configuration, where such a linear combination  $P$  is interpreted as the asserting that  $P = 0$ . As a matter of fact, all known lower bounds for monomial space even hold in the case where arbitrary Boolean functions of clauses are allowed. The term monomial space comes from the fact that this concept captures space in proof systems employing algebraic reasoning.

Namely, seeing clauses as monomials – a clause  $\ell_1 \vee \cdots \vee \ell_r$  is seen as the monomial  $\overline{\ell_1} \cdots \overline{\ell_r}$  – the question of whether a set of clauses over the variables  $x_1, \dots, x_n$  is unsatisfiable, becomes the question of whether the polynomial 1 belongs to the ideal generated by those clauses and the clauses  $x_i^2 - x_i$  and  $x_i + \overline{x_i} - 1$  in  $\mathbb{F}[x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}]$ . A systematic way of generating this ideal, in a space oriented model, is the following [1]. Configurations are sets of polynomials over  $\mathbb{F}[x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}]$ . A refutation of a CNF formula  $F$ , in configurational form, is a sequence  $\mathcal{M}_1, \dots, \mathcal{M}_t$  of configurations where  $\mathcal{M}_1$  is empty,  $\mathcal{M}_t$  contains the empty clause and for  $i > 1$ ,  $\mathcal{M}_i$  is obtained from  $\mathcal{M}_{i-1}$  by one of the following rules:

**Axiom download:**  $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ , where  $C$  is either a clause of  $F$ ,  $x_i^2 - x_i$ , or  $x_i + \overline{x_i} - 1$ .

**Inference:**  $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{P\}$ , where  $P$  is either a linear combination of polynomials in  $\mathcal{M}_{i-1}$  or a literal multiplied by some polynomial in  $\mathcal{M}_{i-1}$ .

**Erasure:**  $\mathcal{M}_i \subseteq \mathcal{M}_{i-1}$ .

The monomial space of such a refutation is the maximum number of distinct monomials occurring in a configuration. The monomial space,  $\text{MSpace}(F \vdash \perp)$ , of  $F$ , is the minimum monomial space over all refutations of  $F$ .

The gap shown in the previous section can be extended to stronger versions of dynamic satisfiability that have been used to show monomial space lower bounds, thus showing a gap between resolution width and monomial space. The configurations in those are not assignments as in Definition 6.1, but sets of assignments. They will not be arbitrary sets however; they will have a certain structure. Namely, we call a set  $H$  of assignments *admissible*, if it is of the form

$$H = H_1 \times \cdots \times H_r \stackrel{\text{def}}{=} \{\alpha_1 \cup \cdots \cup \alpha_r \mid \alpha_i \in H_i\},$$

where each  $H_i$  is a non-empty set of non-empty assignments, for any two assignments  $\alpha_i \in H_i$  and  $\alpha_j \in H_j$  for  $i \neq j$ , the domains of  $\alpha_i$  and  $\alpha_j$  do not intersect, and moreover, if an assignment  $\alpha \in H_i$  gives the value  $\epsilon$  to a variable  $x$ , then there is also an assignment  $\alpha' \in H_i$  giving to  $x$  the value  $1 - \epsilon$ . The  $H_i$ 's are called the *factors* of  $H$ ; we write  $\|H\|$  for their number. We write  $H' \sqsubseteq H$  if every factor of  $H'$  is a factor of  $H$ .

► **Definition 8.1** [11, 12]. Let  $F$  be a CNF formula and let  $k$  be a natural number. We say that  $F$  is *k-extensible* if there is a non-empty set of admissible configurations  $\mathcal{H}$  such that for each  $H \in \mathcal{H}$ ,

1. if  $H' \sqsubseteq H$ , then  $H' \in \mathcal{H}$ ;
2. if  $\|H\| < k$  and  $C$  is a clause of  $F$ , then there is an  $H' \sqsupseteq H$  in  $\mathcal{H}$ , such that every  $\alpha \in H'$  satisfies  $C$ .

► **Theorem 8.1** [11, 12]. *If  $F$  is k-extensible, then  $\text{MSpace}(F \vdash \perp) \geq \lfloor k/4 \rfloor$ .*

Lemma 7.1 with the same proof applies here as well.

► **Lemma 8.1.** *Let  $F$  and  $G$  be CNF formulas over disjoint sets of variables. If  $F$  is k-extensible and  $G$  is  $\ell$ -extensible, then  $F \times G$  is  $(k + \ell)$ -extensible.*

**Proof.** Let  $\mathcal{H}$  and  $\mathcal{I}$  be sets of admissible configurations witnessing the  $k$  and  $\ell$ -extensibility of  $F$  and  $G$ . Since  $F$  and  $G$  are over disjoint sets of variables, we may assume that the domains for any of two assignments in  $\mathcal{H}$  and  $\mathcal{I}$  do not intersect. Set

$$\mathcal{J} := \{H \times I \mid H \in \mathcal{H} \ \& \ I \in \mathcal{I}\}.$$

Clearly,  $\mathcal{J}$  is a set of admissible configurations. We claim that it satisfies the conditions of Definition 8.1 for the parameter  $k + \ell$ . Closure under  $\sqsubseteq$  immediately follows from the fact that  $\mathcal{H}$  and  $\mathcal{I}$  are closed under  $\sqsubseteq$ . For the second condition, suppose that  $J = H \times I \in \mathcal{J}$ ,  $\|J\| < k + \ell$ , and let  $C_i \vee D_j$  be a clause of  $F \times G$ , where  $C_i$  is a clause of  $F$  and  $D_j$  is a clause of  $G$ . Since  $\|J\| < k + \ell$ , either  $\|H\| < k$  or  $\|I\| < \ell$ . In the first case, there is an  $H' \sqsupseteq H$  in  $\mathcal{H}$  such that all assignments in  $H'$  satisfy  $C_i$ . Then  $H' \times I$  is an admissible configuration in  $\mathcal{J}$  such that all assignments in it satisfy  $C_i \vee D_j$ . The second case is analogous. ◀

Therefore, we get:

► **Theorem 8.2.** *There are CNF formulas  $G$  with  $n^2$  variables, size  $O(n)^n$ , and such that  $W_R(G \vdash \perp) = O(n)$  and  $\text{MSpace}(G \vdash \perp) = \Omega(n^2)$ .*

**Proof.** Again, let  $F$  be a CNF formula with  $n$  variables, width  $O(1)$  and size  $\Theta(n)$ , that is  $\Omega(n)$ -extensible. Such formulas exist, see [11, 16, 12, 9]. The formulas  $F^n := F_1 \times \cdots \times F_n$ , where the  $F_i$ 's are copies of  $F$  over mutually disjoint sets of variables, have resolution width  $O(n)$ , and from Lemma 8.1 they are  $\Omega(n^2)$ -extensible, thus from Theorem 8.1 require  $\Omega(n^2)$  monomial space. ◀

## 9 A super-polynomial separation between resolution and $LK^-$ size

Many of the relations in resolution involving width, can be as well stated for  $LK^-$ . In fact, they seem to be better suited for  $LK^-$ ; there, the additive  $W(F)$  factor that naturally comes with resolution width disappears. We have already seen that  $W_{LK^-}(F \vdash \perp) \leq \text{CSpace}(F \vdash \perp)$ , refining the relation between clause space and width of [4]. But let us give an alternative, constructive proof, here. For sets  $S$  and  $T$  of formulas, we write  $S \models T$  if every total assignment satisfying every formula in  $S$ , also satisfies every formula in  $T$ .

► **Theorem 9.1.** *For any unsatisfiable CNF formula  $F$ ,  $W_{LK^-}(F \vdash \perp) \leq \text{CSpace}(F \vdash \perp)$ .*

**Proof.** Let  $\mathcal{M}_1, \dots, \mathcal{M}_t$  be a refutation of  $F$ , of clause space  $s$ . We shall construct a sequence  $\mathbf{T}_1, \dots, \mathbf{T}_t$  of trees, the vertices of which are labelled by sets of literals, such that for every set  $S$  labelling a leaf of  $\mathbf{T}_i$ ,  $S \models \mathcal{M}_i$  and  $|S| \leq |\mathcal{M}_i|$ .

We set  $\mathbf{T}_1$  to be a tree with one vertex labelled by the empty set. Now, suppose we have constructed  $\mathbf{T}_{i-1}$ . If  $\mathcal{M}_i$  results from  $\mathcal{M}_{i-1}$  via an inference step, we set  $\mathbf{T}_i := \mathbf{T}_{i-1}$ . If  $\mathcal{M}_i \subseteq \mathcal{M}_{i-1}$ , then we add to every leaf of  $\mathbf{T}_{i-1}$  labelled by a satisfiable set  $S$ , a child labelled by a subset  $S' \subseteq S$  such that  $|S'| \leq |\mathcal{M}_i|$  and  $S' \models \mathcal{M}_i$ . Finally, if  $\mathcal{M}_i = \mathcal{M}_{i-1} \cup \{C\}$ , for a clause  $C = \ell_1 \vee \dots \vee \ell_r$  of  $F$ , we add to every child of  $\mathbf{T}_i$  labelled by a satisfiable set  $S$ ,  $r$  children labelled by the sets  $S \cup \{\ell_j\}$ .

Replacing each set  $\{\ell_1, \dots, \ell_k\}$  occurring in  $\mathbf{T}_t$ , by the clause  $\bar{\ell}_1 \vee \dots \vee \bar{\ell}_k$ , we get an  $LK^-$  refutation of  $F$  of width at most  $s$ . It is clear from the construction of  $\mathbf{T}_t$  that every clause has width at most  $s$  and every clause not at a leaf, results from the clauses at its children via either the weakening or the  $\forall L$  rule. Moreover, since  $\mathcal{M}_t$  is unsatisfiable, no set labelling a leaf of  $\mathbf{T}_i$  is satisfiable, that is sets at the leaves become weakenings of axioms. ◀

Next, we have the size-width relations of [8]. Here the proofs are the same as those in [8].

► **Theorem 9.2.** *For any unsatisfiable CNF formula  $F$ ,  $W_{LK^-}(F \vdash \perp) \leq \log S_{\text{TR}}(F \vdash \perp) + 1$ .*

**Proof.** This is in fact a weakened version of Theorem 9.1, as  $\text{CSpace}(F \vdash \perp) \leq \log S_{\text{TR}}(F \vdash \perp) + 1$  [15]. But let us give a direct construction instead. We shall construct, by induction on  $s$ , for every tree-like resolution refutation  $\mathbf{T}$  of  $F$  of size  $s$ , an  $LK^-$  refutation of  $F$  of width at most  $\log s + 1$ .

If  $\mathbf{T}$  has size 1, then it has width 0; if it has size 3 then it has width 2. For the inductive step, suppose that  $\mathbf{T}$  has size  $s > 3$ . Let  $\mathbf{T}_1$  and  $\mathbf{T}_2$  be the subproofs of  $\mathbf{T}$ , deriving  $\neg x$  and  $x$  respectively, for some variable  $x$ . One of  $\mathbf{T}_1$  and  $\mathbf{T}_2$ , say  $\mathbf{T}_1$ , must have size at most  $s/2$ .  $\mathbf{T}_1|_x$  is a refutation of  $F|_x$  of size at most  $s/2$ , and  $\mathbf{T}_2|_{\bar{x}}$  is a refutation of  $F|_{\bar{x}}$  of size less than  $s$ . From the induction hypothesis, there are  $LK^-$  refutations  $\pi_1$  and  $\pi_2$  of  $F|_x$  and  $F|_{\bar{x}}$  of width  $\log s$  and  $\log s + 1$  respectively. Start with  $\pi_2$ . To every application of Rule (5.1), add the extra premise  $C \vee x$ . But  $x$  can be derived from  $\pi_1$ , and hence all those  $C \vee x$  can be derived via the weakening rule from  $x$ : We can do the same to  $\pi_1$ , now adding  $C \vee \bar{x}$  as the extra premise, and moreover add to every clause the variable  $x$ . The refutation obtained when we combine  $\pi_1$  and  $\pi_2$  is a valid  $LK^-$  refutation of  $F$  of width at most  $\log s + 1$ . ◀

► **Theorem 9.3.** *For any unsatisfiable CNF formula  $F$  over  $n$  variables,  $W_{LK^-}(F \vdash \perp) = O\left(\sqrt{n \log S_{LK^-}(F \vdash \perp)}\right)$ .*

**Proof.** The construction will be the same as that of Theorem 9.2. The problem here is that it is not clear what variable to choose to recurse on. The trick is to choose the variable that appears more often in the clauses of the proof.

For an  $\text{LK}^-$  refutation  $\pi$  of a CNF and a  $d \geq 0$ , let  $\pi^*$  be the set of clauses in  $\pi$  of width greater than  $d$ . We call the clauses in  $\pi^*$  the *fat* clauses of  $\pi$ . We show, by induction on  $n$ , that for any CNF  $F$  in  $n$  variables, any  $\text{LK}^-$  refutation  $\pi$  of  $F$  and any integers  $d, b \geq 0$ ,

$$|\pi^*| < a^b \implies W_{\text{LK}^-}(F \vdash \perp) \leq d + b,$$

where  $a = (1 - d/(2n))^{-1}$ . The theorem follows taking  $\pi$  to be of minimum size and  $d := \lceil \sqrt{n \log S(\pi)} \rceil$ .

If  $n = 1$ , then there is an  $\text{LK}^-$  refutation of  $F$  of width 2, and in the case that  $b + d < 2$  the implication becomes trivially true. For the inductive step, suppose that  $n > 1$ , let  $d, b \geq 0$ , and let  $\pi$  be an  $\text{LK}^-$  refutation of  $F$  with  $|\pi^*| < a^b$ . If  $b = 0$ , then  $\pi$  itself is a refutation of width at most  $d + b$ . Suppose  $b > 0$ . There are  $2n$  literals, so there must be some literal, say the variable  $x$ , appearing in at least  $d|\pi^*|/(2n)$  clauses in  $\pi^*$ .  $\pi|_x$  is an  $\text{LK}^-$  refutation of  $F|_x$  with at most  $|\pi^*|(1 - d/(2n)) < a^{b-1}$  fat clauses, so by the induction hypothesis (notice that  $a$  is a decreasing function of  $n$ ), there is an  $\text{LK}^-$  refutation  $\pi'$  of  $F|_x$  of width at most  $d + b - 1$ . Furthermore,  $\pi|_{\bar{x}}$  is an  $\text{LK}^-$  refutation of  $F|_{\bar{x}}$  with less than  $a^b$  clauses, so by the induction hypothesis, there is an  $\text{LK}^-$  refutation  $\pi''$  of  $F|_{\bar{x}}$  of width at most  $d + b$ . Combining  $\pi'$  and  $\pi''$  as in the proof of Theorem 9.2, we get an  $\text{LK}^-$  refutation of  $F$  of width at most  $d + b$ . ◀

Notice that in Theorem 9.2 and the relation  $W_{\text{LK}^-}(F \vdash \perp) \leq \text{CSpace}(F \vdash \perp)$ , we have  $\text{LK}^-$  in the left hand side and resolution in the right hand side. That is to say, cuts are eliminated when constructing the small width proofs. It is tempting to speculate on whether the same is also true for Theorem 9.3, that is whether we can replace  $S_{\text{LK}^-}(F \vdash \perp)$  with  $S_R(F \vdash \perp)$ . After all, the only place in the proof of Theorem 9.3 where we need  $\text{LK}^-$  in the right hand side is the case  $b = 0$ . Theorem 7.1 says that this cannot be true. In fact, the formulas of Theorem 7.1 give the main theorem of this section, which is:

► **Theorem 9.4.** *There is a CNF formula  $F$  with  $n^2$  variables and size  $O(n)^n$ , such that  $S_R(F \vdash \perp) = O(n)^n$  but  $S_{\text{LK}^-}(F \vdash \perp) \geq \exp(\Omega(n^2))$ .*

**Proof.** The formulas of Theorem 7.1 are such. The upper bound  $S_R(F \vdash \perp) = O(n)^n$  follows from the construction of Lemma 7.2. The lower bound follows from the fact that  $W_{\text{LK}^-}(F \vdash \perp) = \Omega(n^2)$  and Theorem 9.3. Namely, Theorem 9.3 gives

$$S_{\text{LK}^-}(F \vdash \perp) \geq \exp\left(\Omega\left(\frac{(W_{\text{LK}^-}(F \vdash \perp))^2}{n^2}\right)\right) = \exp(\Omega(n^2)). \quad \blacktriangleleft$$

It is important to note that the  $\exp(\Omega(n^2))$  lower bound in Theorem 9.4 holds for the version of  $\text{LK}^-$  operating on clauses, where the clauses of the CNF formula  $F$  to be refuted are viewed as disjunctions of unbounded arity. It does not hold when the clauses of  $F$  are made up from binary disjunctions and moreover we are free to choose the order in which they are applied. If  $\vee$  in the definition of  $F \times G$ , is seen as a binary disjunction, then having derived  $C_1, \dots, C_n \rightarrow$  and  $D_1, \dots, D_t \rightarrow$ , it is easy to see that we may derive from these sequents  $C_1 \vee D_1, \dots, C_1 \vee D_t, \dots, C_s \vee D_1, \dots, C_s \vee D_t \rightarrow$  in  $s \cdot t$  steps, and in this case  $F^n$  in Theorem 9.4 has an  $\text{LK}^-$  refutation of size  $n^{O(n)}$ . An analogous situation occurs between the tree-like versions of  $\text{LK}^-$  and resolution [3]. But let us notice, concluding, that with binary disjunctions,  $\text{LK}^-$  cannot be seen as a system operating on clauses, and it becomes rather unnatural to compare it with resolution – it is not even clear, in this case, whether resolution can polynomially simulate  $\text{LK}^-$ .  $\text{LK}^-$  for clauses consisting of binary disjunctions is closer to resolution with limited extension, in which case resolution does polynomially simulate it [24].

## 10 Conclusion

We showed a quadratic gap between resolution and cut-free sequent calculus width. In terms of the sequent calculus, this says that atomic cuts can shorten the width of proofs. It is well known that cuts can make proofs exponentially shorter. Allowing arbitrary cuts we get a system polynomially equivalent with any Frege system. These are very powerful; proving non-trivial lower bounds for them is completely out of reach of current methods. But even allowing cuts of depth  $d + 1$  in an LK system that has cuts of depth  $d$  for any constant  $d \geq 0$ , gives exponentially shorter proofs [19]. And this goes lower: For any constant  $k \geq 0$ , allowing as cut formulas conjunctions and disjunctions of arity  $k + 1$  in an LK system that has as cuts conjunctions and disjunctions of arity at most  $k$ , again gives exponentially shorter proofs [22]. We show in this paper that even allowing propositional variables as cuts, gives super-polynomially shorter proofs.

Cut-free sequent width for refuting CNF formulas naturally compares to well studied complexity measures related to resolution: it sits between resolution width and clause space. Our quadratic gap in particular, provides a separation between resolution width and clause space. Stronger such separations are known [6, 7]. Nonetheless, our basic construction extends to provide a quadratic gap between resolution width and monomial space. This is to be seen in conjunction with relation (1.2) showing that monomial space provides an upper bound to resolution width.

Several questions remain open:

1. Can cut-free sequent calculus width for refuting CNF formulas be bounded in terms of resolution width? Given the similarity between the two measures, the combination of Lemmas 7.1 and 7.2 giving a quadratic separation might come as a surprise. Can this separation be improved? A strong separation in particular, would give an exponential separation between resolution and cut-free sequent calculus.
2. Our super-polynomial separation of resolution and cut-free sequent calculus on the one hand applies only when clauses are seen as disjunctions of unbounded arity. On the other hand, it involves formulas whose size grows exponentially. Can there be a separation independent of the representation of clauses? Can there be a separation for formulas whose size grows polynomially?
3. Cut-free sequent calculus width is bounded by clause space. Can it be bounded in terms of monomial space in a relation similar to (1.2)? This is a good point to also mention that whether (1.2) can be improved to a linear inequality or there are examples where it is tight is unknown as well, and there do not seem to be strong indications for which case is true.
4. We show that resolution width and monomial space cannot coincide. Whether they coincide up to polynomial factors however remains open, although it is speculated (cf. [18]) that this is not the case, and moreover, as it is the case for resolution width and clause space [6, 7], there is an  $O(1)$  vs  $\Omega(n/\log n)$  separation.

---

## References

- 1 Michael Alekhovich, Eli Ben-Sasson, Alexander Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM J. of Computing*, 31:1184–1211, 2002.
- 2 Noriko Arai. Relative efficiency of propositional proof systems: resolution vs. cut-free LK. *Annals of Pure and Applied Logic*, 104:3–16, 2000.
- 3 Noriko Arai, Toniann Pitassi, and Alasdair Urquhart. The complexity of analytic tableaux. *J. of Symbolic Logic*, 71:777–790, 2006.

- 4 Albert Atserias and Victor Dalmau. A combinatorial characterization of resolution width. *J. of Computer and System Sciences*, 74:323–334, 2008.
- 5 Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures & Algorithms*, 23:92–109, 2003.
- 6 Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Pr. of the 49th Annual IEEE Symp. on Foundations of Computer Science*, pages 709–718, 2008.
- 7 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Pr. of the 2nd Symp. on Innovations in Computer Science*, pages 401–416, 2011.
- 8 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *J. of the ACM*, 48:149–169, 2001.
- 9 Patrick Bennett, Ilario Bonacina, Nicola Galesi, Tony Huynh, Mike Molloy, and Paul Wollan. Space proof complexity for random 3-cnfs. *Information and Computation*, 255:165–176, 2017.
- 10 Ilario Bonacina. Total space in resolution is at least width squared. In *Pr. of the 43rd International Colloquium on Automata, Languages, and Programming*, pages 56:1–56:13, 2016.
- 11 Ilario Bonacina and Nicola Galesi. Pseudo-partitions, transversality and locality: a combinatorial characterization for the space measure in algebraic proof systems. In *Pr. of the 4th Conf. on Innovations in Theoretical Computer Science*, pages 455–472, 2013.
- 12 Ilario Bonacina and Nicola Galesi. A framework for space complexity in algebraic proof systems. *J. of the ACM*, 62:23:1–23:20, 2015.
- 13 Stephen Cook and Robert Reckhow. On the lengths of proofs in the propositional calculus (pr. ver.). In *Pr. of the 6th Annual ACM Symp. on Theory of Computing*, pages 135–148, 1974.
- 14 Juan Luis Esteban, Nicola Galesi, and Jochen Messner. On the complexity of resolution with bounded conjunctions. *Theoretical Computer Science*, 321:347–370, 2004.
- 15 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171:84–97, 2001.
- 16 Yuval Filmus, Massimo Lauria, Mladen Miksa, Jakob Nordström, and Marc Vinyals. Towards an understanding of polynomial calculus: New separations and lower bounds (extended abstract). In *Pr. of the 40th International Colloquium on Automata, Languages, and Programming*, pages 437–448, 2013.
- 17 Nicola Galesi, Leszek Kołodziejczyk, and Neil Thapen. Polynomial calculus space and resolution width. In *Pr. of the 60th Annual IEEE Symp. on Foundations of Computer Science*, pages 1325–1337, 2019.
- 18 Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: amplifying communication complexity hardness to time-space trade-offs in proof complexity. In *Pr. of the 44th Annual ACM Symp. on Theory of Computing*, pages 233–248, 2012.
- 19 Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *J. of Symbolic Logic*, 59:73–86, 1994.
- 20 Theodoros Papamakarios and Alexander Razborov. Space characterizations of complexity measures and size-space trade-offs in propositional proof systems. *J. of Computer and System Sciences*, 137:20–36, 2023.
- 21 Robert Reckhow. *On the lengths of proofs in the propositional calculus*. PhD thesis, Department of Computer Science, University of Toronto, 1975.
- 22 Nathan Segerlind, Samuel Buss, and Russell Impagliazzo. A switching lemma for small restrictions and lower bounds for k-DNF resolution. *SIAM J. of Computing*, 33:1171–1200, 2004.
- 23 Raymond Smullyan. *First-order Logic*. Dover, 1995.
- 24 Grigori Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic, part 2*, pages 115–125, 1968.
- 25 Alasdair Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1:425–467, 1995.



# Realizing Finitely Presented Groups as Projective Fundamental Groups of SFTs

Léo Paviet Salomon ✉

Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

Pascal Vanier ✉ 🏠 

Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

---

## Abstract

Subshifts are sets of colourings – or tilings – of the plane, defined by local constraints. Historically introduced as discretizations of continuous dynamical systems, they are also heavily related to computability theory. In this article, we study a conjugacy invariant for subshifts, known as the projective fundamental group. It is defined via paths inside and between configurations. We show that any finitely presented group can be realized as a projective fundamental group of some SFT.

**2012 ACM Subject Classification** Theory of computation → Models of computation; Mathematics of computing → Discrete mathematics

**Keywords and phrases** Subshifts, Wang tiles, Dynamical Systems, Computability, Subshift of Finite Type, Fundamental Group

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.75

**Related Version** *Full Version:* <https://hal.science/hal-03622497>

**Funding** This research was partially funded by ANR JCJC 2019 19-CE48-0007-01.

**Acknowledgements** The authors would like to thank all the anonymous referees of this paper for their remarks which helped improve the exposition.

## 1 Introduction

A  $d$ -dimensional subshift is a set of colourings of  $\mathbb{Z}^d$  by a finite number of colours which avoid some family of forbidden patterns. If the family is finite, it is called a subshift of finite type (SFT). Most problems concerning subshifts in dimension  $d \geq 2$  are undecidable [6, 20, 19], due to the fact that sets of Wang tilings are SFTs.

Together with the shift action  $\sigma$ , a subshift forms a dynamical system. Interesting dynamical aspects are usually invariant by conjugacy, which is the isomorphism notion for subshifts. Most conjugacy invariants of subshifts in dimensions  $d \geq 2$  are linked to computability theory or complexity theory. Historically, the first example was the characterization of the topological entropies of multi-dimensional SFTs as the upper semi-computable numbers [25]. Afterwards, many other computational characterizations of conjugacy invariants have been obtained: growth-type invariants [30], subactions [23, 4, 14] and so on.

Links between groups and subshifts have recently seen a surge in interest with several different approaches: subshifts can be defined on groups instead of  $\mathbb{Z}^d$  [1, 3] and some properties of the group are linked to decidability questions on the subshifts on it [26, 2, 15]. Analogies between groups and subshifts have allowed new characterizations to be proved for subshifts [27].

Another avenue is to associate a group to a subshift in order to construct conjugacy invariants in several ways [29, 22, 17]. The most well-known such group is the automorphism group, which is still not very well understood: for instance, while it is known that SFTs with positive entropy have very complex automorphism groups [24] or that SFTs whose



© Léo Paviet Salomon and Pascal Vanier;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 75; pp. 75:1–75:15

Leibniz International Proceedings in Informatics



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

automorphism group has undecidable word problem can be constructed [18], it is still not known whether the automorphism groups of the full shifts on 2 and 3 symbols are the same. Apart from the low complexity setting [13, 12] not much is understood about it.

In this article, we study another group-related conjugacy invariant called the *projective fundamental group* introduced by Geller and Propp [17]. Fundamental groups are an object of interest in several fields of theoretical computer science, in particular graph reconfigurations [37], which bear links with a particular class of subshifts called *hom-shifts* [10] which are defined with a graph of allowed adjacency of colours. These are subshifts with a computable language that still exhibit interesting behavior [16]. An essential tool in their study is their *universal cover*, a graph which has strong ties to their projective fundamental group. Fundamental groups are also of interest when studying the “defects” in tilings [33, 5], or obstruction to the tileability of finite, untiled “holes” in tilings [11, 36]. In particular, provided that an SFT satisfies some mixing-like hypothesis, there is an explicit link between its fundamental cocycles [35, 34] and its projective fundamental group.

In the usual topological setting (see for example [21]), the fundamental group  $\pi_1(X)$  of a space  $X$  is a topological invariant which describes the number of holes and the general shape of  $X$ . It is defined as the group of equivalence classes of loops through continuous deformation, together with the composition operation. In this setting, the fundamental group is well-defined only when  $X$  is path-connected.

When viewed as subspaces of the Cantor space, subshifts are totally disconnected. Nevertheless, one can still define a notion of projective fundamental group using paths and deformations (see Subsection 3.1 for details). As in the classical setting, this notion is only well-defined in the case of *projectively connected subshifts*, the appropriate notion of path-connectedness. This property resembles mixing properties (see for instance [9] or [32]), but it is not known whether any of the mixing properties defined in [9] imply projective connectedness of an SFT, although some partial results exist [33, 35]. Projective connectedness is undecidable but we do not know how hard: it is open whether it belongs to the arithmetical hierarchy.

As a conjugacy invariant, the fundamental group allows one to distinguish between some subshifts which share the same entropy and periodicity data. It is also better understood than the automorphism group in the sense that the authors in [17] explicitly compute it for several well-known subshifts: the full shifts on any alphabet always have trivial fundamental group, the square-ice has  $\mathbb{Z}$  and  $k$ -to-1 factors of full shifts – *i.e.* in which every point has exactly  $k$  preimages by the factor map – always have a fundamental group with finite order  $k$ . They also prove that any group of finite order is realizable as a fundamental group of some SFT.

The main result of this article is that any finitely presented group can be the fundamental group of an SFT:

► **Theorem 1.** *Let  $G = \langle S \mid R \rangle$  be a finitely presented group. Then, there is a subshift of finite type  $X$  satisfying:*

- *$X$  is projectively connected,*
- *the projective fundamental group of  $X$  is isomorphic to  $G$ .*

We do not think that this constitutes a characterization of projective fundamental groups of SFTs, as we do not have a matching upper bound on the hardness its word problem. However, this theorem implies that the hardness of the word problem of the fundamental group – *i.e.* given a SFT, decide the word problem of its fundamental group – can be any recursively enumerable degree [8], and in particular that its upper bound is at least  $\Sigma_1^0$ -hard [31, 7]. It also implies that any undecidable property on finitely presented groups is undecidable for projective fundamental groups.

The main construction of the paper is quite different from other constructions used in undecidability results on tilings and subshifts: it does not use an aperiodic subshift.

The paper is organized as follows. After recalling the symbolic dynamics background in Section 2, we introduce the projective fundamental group in Subsection 3.1, some examples in Subsection 3.2 and finally in Section 4 we prove Theorem 1.

## 2 Definitions

A **d-dimensional full shift** on some finite alphabet  $\Sigma$  is the set  $\Sigma^{\mathbb{Z}^d}$ , together with the **shift-actions**  $\sigma_{\mathbf{u}}: \Sigma^{\mathbb{Z}^d} \rightarrow \Sigma^{\mathbb{Z}^d}$  defined for  $\mathbf{u} \in \mathbb{Z}^d$  by  $\sigma_{\mathbf{u}}(x)(\mathbf{v}) = x(\mathbf{u} + \mathbf{v}) = x_{\mathbf{u}+\mathbf{v}}$ . The underlying topology is the one induced by the **Cantor distance**, defined on  $\Sigma^{\mathbb{Z}^d}$  by

$$d(x, y) = 2^{-\min\{\|\mathbf{u}\|_{\infty} \mid x_{\mathbf{u}} \neq y_{\mathbf{u}}\}},$$

Two configurations are close in this topology if they agree on a large central square. A **subshift** is a closed, shift-invariant subset of some full shift. We call **configurations** of a subshift  $X$  the points of  $X$ .

Alternatively, subshifts can be defined using forbidden patterns. We call **pattern** any element  $P \in \Sigma^U$  where  $U \subset \mathbb{Z}^d$  is finite and is the **support** of  $P$ , denoted by  $\text{supp}(P)$ . For a configuration  $x$ , we say that  $P$  appears in  $x$  if there exists  $\mathbf{u} \in \mathbb{Z}^d$  such that  $\sigma_{\mathbf{u}}(x)|_U = P$ . Let  $\mathcal{F}$  be a collection (finite or not) of patterns. Then the set

$$X_{\mathcal{F}} = \left\{ x \in \Sigma^{\mathbb{Z}^d} \mid \forall P \in \mathcal{F}, P \text{ does not appear in } x \right\}$$

is a subshift. In fact, for any subshift  $X$ , there exists a family of patterns  $\mathcal{F}$  such that  $X = X_{\mathcal{F}}$ . A subshift  $X$  is a **subshift of finite type (SFT)** if there exists a finite  $\mathcal{F}$  such that  $X = X_{\mathcal{F}}$ .

For a given subshift  $X$  defined by a fixed family of forbidden patterns  $\mathcal{F}$ , a pattern  $P \in \Sigma^U$  is **locally admissible** if it contains no forbidden patterns  $F \in \mathcal{F}$ . It is **globally admissible** or **extensible** if it appears in some configuration  $x \in X$ .

## 3 Projective Fundamental Group

### 3.1 Intuitions and definitions

The Projective Fundamental Group, introduced by Geller and Propp [17], resembles the usual fundamental group construction in the topological setting: it is defined through paths, loops, and a homotopy notion. However, instead of directly considering paths between points of the subshift, they are defined between finite patterns with the same support. By doing so, one actually constructs a family of – potentially different – fundamental groups, for each finite support  $B \subset \mathbb{Z}^2$ . In order to obtain a single group, the projective fundamental group, one takes their inverse (also known as projective) limit. We will construct a subshift by defining a set  $\mathcal{T}$  of tiles. A configuration will then be a mapping  $x: \mathbb{Z}^2 \rightarrow \mathcal{T}$  associating a tile to each point of the plane and which verifies some adjacency rules depending on  $\mathcal{T}$ . Contrary to the usual convention, we will consider that when embedding such a configuration in the Euclidean plane  $\mathbb{R}^2$ , the tile in position  $(i, j)$  is a unit square whose bottom-left corner is placed on  $(i, j)$ , as opposed to its center. This is merely a discussion about conventions, but it will make some definitions substantially simpler.

Fix a support  $B \subset \mathbb{Z}^2$ . In what follows  $B$  will be called an **aperture window**. Most of the time, we will restrict ourselves to the windows  $B_n = \llbracket -n, n-1 \rrbracket^2$ . We choose this asymmetrical window to simplify some definitions, but also for consistency with the

aforementioned convention. In any configuration  $x$ , the tile  $x_{(0,0)}$  in position  $(0,0)$  will therefore be seen as the square whose bottom-left (resp. top-right) corner is  $(0,0)$  (resp.  $(1,1)$ ).

Consider  $P, P'$  two extensible patterns of support  $B$  and two points of the grid  $\mathbf{v}, \mathbf{v}' \in \mathbb{Z}^2$ . A **path** between  $(P, \mathbf{v})$  and  $(P', \mathbf{v}')$  is a sequence of pairs of patterns and of points of  $\mathbb{Z}^2$  (or equivalently, two sequences of the same length). The sequence of points represents an actual, “geometric” path, called its **trajectory**, that is to say a sequence of vertices of  $\mathbb{Z}^2$  starting at  $\mathbf{v}$  and ending at  $\mathbf{v}'$ , where consecutive vertices are at euclidean distance exactly 1. The sequence of patterns associates with each one of those vertices  $\mathbf{v}_t$  a pattern  $P_t$ , that needs to be coherent with the path: when moving to the next vertex  $\mathbf{v}_{t+1}$  on the trajectory, the next pattern  $P_{t+1}$  needs to be coherent with  $P_t$ , that is to say, they should be equal where their supports overlap (see Definition 2 for a precise statement). For example, in the full shift over two symbols  $\{0, 1\}$ , and for  $B = B_1$ , take the following patterns:

$$P_1 = \left( \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, (0,0) \right), \quad P_2 = \left( \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 0 & 1 \\ \hline \end{array}, (1,0) \right), \quad P_3 = \left( \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array}, (1,0) \right)$$

The tile in position  $(0,0)$  is represented in red. The sequence  $(P_1, P_2)$  is a valid path, as the overlapping parts of the support are equal in both patterns, but  $(P_1, P_3)$  is not because the point  $(0,0)$  is tiled by 0 in the first pattern but by 1 in the second one. Moreover, the pattern obtained by “merging” two consecutive patterns also needs to be an extensible pattern.

**► Definition 2 (Path).** Let  $B \subset \mathbb{Z}^2$  be a finite set, a path of **aperture window**  $B$  is a finite sequence  $(P_t, \mathbf{v}_t)_{0 \leq t \leq N}$  such that for any  $t$  with  $0 \leq t \leq N$ :

- $P_t$  is an extensible pattern of  $X$  of support  $B + \mathbf{v}_t$ ,
- $\mathbf{v}_t$  is adjacent to  $\mathbf{v}_{t+1}$ , i.e.,  $\mathbf{d}_t = \mathbf{v}_{t+1} - \mathbf{v}_t$  has euclidean norm exactly 1,
- $P_t(u) = P_{t+1}(u)$  for any  $u \in B \cap \sigma_{\mathbf{d}_t}(B)$ , i.e., consecutive patterns overlap,
- the pattern  $P_t \cup P_{t+1}$  obtained by merging  $P_t$  and  $P_{t+1}$  is extensible in  $X$ .

The first and last element of the sequence are respectively called the **starting point** and the **ending point** of the path. If they are equal, the path is called a **loop**. The path  $(P_{N-t}, \mathbf{v}_{N-t})_{0 \leq t \leq N}$  is called its **inverse path**. If  $p$  is a path, its inverse will be denoted by  $p^{-1}$

The sequence  $(\mathbf{v}_t)_{0 \leq t \leq N}$  is called the **trajectory** of the path.

Two paths may be composed when the first one ends where the second one starts:

**► Definition 3 (Path composition).** Given  $p = (P_t, \mathbf{v}_t)_{0 \leq t \leq N}$  and  $p' = (P'_t, \mathbf{v}'_t)_{0 \leq t \leq N'}$  two paths such that  $(P_N, \mathbf{v}_N) = (P'_0, \mathbf{v}'_0)$  we denote by  $p * p'$  the path

$$p * p' = (P_0, \mathbf{v}_0) \dots (P_N, \mathbf{v}_N) (P'_1, \mathbf{v}'_1) \dots (P'_{N'}, \mathbf{v}'_{N'}).$$

**► Definition 4 (Coherent path).** A path  $p = (P_i, \mathbf{v}_i)_{i \leq N}$  is coherent if all its patterns are equal on the points where their supports overlap, and furthermore, the pattern obtained by merging all the  $P_i$  is globally admissible in  $X$ . In that case, for any  $x \in X$  containing  $\bigcup_{i \leq N} P_i$ , we say that  $p$  can be **traced** in  $x$ .

**► Definition 5 (Coherent path decomposition).** A **coherent decomposition** of a path  $p$  is a sequence  $p_1, \dots, p_L$  of coherent paths such that  $p = p_1 * p_2 \dots * p_L$ , and  $L$  is called the **length** of the decomposition.

One can now define a corresponding homotopy notion: let  $p = p_1 * p_2 * p_3$  be a path and suppose that  $p_2$  can be traced in a single configuration  $x \in X$ . Then, for any  $p'_2$  traced in  $x$  with the same starting and ending point as  $p_2$ , the path  $p_1 * p'_2 * p_3$  is called an **elementary deformation** of  $p$ . As paths might consist of a single point, they can be deformed by inserting or removing loops traced in a single configuration at any step.

► **Definition 6** (Homotopy). *Two paths  $p, p'$  are said to be **homotopic** if there exists a finite sequence of elementary deformations from  $p$  to  $p'$ . This defines an equivalence relation between paths, and we denote by  $[p]$  the equivalence class of  $p$ . If  $p$  and  $p'$  are paths with an aperture window  $B \subset \mathbb{Z}^2$ , we denote by  $p \sim_B p'$  the fact that they are homotopic.*

► **Remark 7.** When two paths are homotopic, they necessarily have the same starting and ending points. When  $B$  is clear from the context, we will simply write  $p \sim p'$ .

With this definition of a path and of homotopy, we can define a fundamental group for each possible aperture window  $B \subset \mathbb{Z}^2$ .

► **Definition 8** (Fundamental Group). *Let  $X$  be a SFT,  $B \subset \mathbb{Z}^2$  an aperture window,  $x_0 \in X$  and  $\mathbf{v} \in \mathbb{Z}^2$ . The **fundamental group** of  $X$  based at  $(x_0, \mathbf{v})$  for the aperture window  $B$ , denoted by  $\pi_1^B(X, (x_0, \mathbf{v}))$ , is the group of all the equivalence classes of loops starting and ending at  $(x_0|_B, \mathbf{v})$  for the homotopy equivalence relation, along with the  $*$  operation.*

Although our paths follow the  $\mathbb{Z}^2$  grid and seem to be discrete and combinatorial objects, it is legitimate to refer to those objects as *homotopy* and *deformations*, which usually suppose some kind of continuity. In fact, this simplification does not entail any loss of generality, compared to paths drawn in  $\mathbb{R}^2$ , and subshifts seen as  $\mathbb{Z}^2$ -invariants subsets of  $\Sigma^{\mathbb{R}^2}$  (see [17, Subshifts and albums] for more details). In order to obtain a single object associated with the subshift, we get rid of this reference to an aperture window by considering the projective limit of those groups to define the **Projective Fundamental Group** of the subshift.

► **Definition 9** (Restriction maps). *For any  $B' \subseteq B \subset \mathbb{Z}^2$ , the map*

$$\begin{aligned} \text{restr}_{B, B'} : \Sigma^B &\rightarrow \Sigma^{B'} \\ P &\mapsto (i \in B' \mapsto P(i)) \end{aligned}$$

*is called the **canonical restriction map** from  $B$  to  $B'$ . We can naturally extend it to  $\bigcup_{\mathbf{v} \in \mathbb{Z}^2} \Sigma^{B+\mathbf{v}}$  so that  $\text{supp}(P) = B + \mathbf{v} \implies \text{supp}(\text{restr}_{B, B'}(P)) = B' + \mathbf{v}$ .*

Intuitively, these maps simply “forget” some parts of the pattern. We also extend these maps to paths: if  $B' \subseteq B$ , the image of a path  $p$  with aperture window  $B$  is a path with the same trajectory with aperture window  $B'$ , obtained by mapping  $\text{restr}_{B, B'}$  element-wise on  $p$ .

► **Definition 10** (Projective path class). *Let  $x, x' \in X$  and  $\mathbf{v}, \mathbf{v}' \in \mathbb{Z}^2$ . A **projective path class** between  $(x, \mathbf{v})$  and  $(x', \mathbf{v}')$  is a sequence  $([p_n])_{n>0}$  along with the canonical restriction maps, such that  $p_n$  is a path of aperture window  $B_n$  between  $(x_{B_n}, \mathbf{v})$  and  $(x'_{B_n}, \mathbf{v}')$ , and for each  $n > n' > 0$ ,  $\text{restr}_{B_n, B_{n'}}(p_n) \sim_{B_{n'}} p_{n'}$ .*

*In the case where  $(x, \mathbf{v}) = (x', \mathbf{v}')$ , we instead say that  $([p_n])_{n>0}$  is a projective loop class based at  $(x, \mathbf{v})$ .*

► **Definition 11** (Projectively connected subshift). *A subshift  $X$  is **projectively connected** if for any two points  $x, x' \in X$ , there exists a projective path class between  $(x, (0, 0))$  and  $(x', (0, 0))$ .*

As before, projective loop classes based at the same  $(x, \mathbf{v})$  can be concatenated component-wise, to obtain another projective loop class.

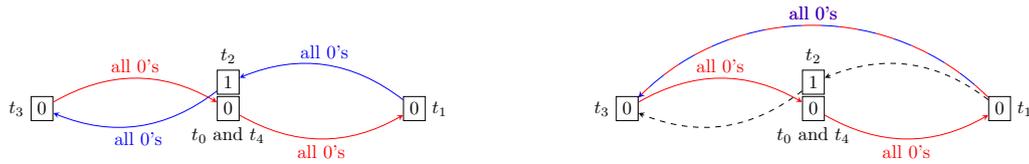
► **Definition 12** (Projective Fundamental Group). *The **projective fundamental group** based at the point  $(x_0, \mathbf{v}) \in X \times \mathbb{Z}^2$  of a subshift  $X$  is the group of projective loop classes based at  $(x_0, \mathbf{v})$ , with the group operation being the component-wise concatenation of projective loop*

classes, and is denoted by  $\pi_1^{proj}(X, (x_0, \mathbf{v}))$ . If  $X$  is projectively connected, then its projective fundamental group does not depend on the chosen basepoint  $(x_0, \mathbf{v})$ , and we denote it by  $\pi_1^{proj}(X)$ .

This is a usual construction of what is called a projective (or inverse) limit in category theory. However, we do not use general properties of inverse limits in the rest of the article.

### 3.2 First example

We slightly modify an example of [17]. Consider the two-dimensional subshift  $X$  on the alphabet  $\{0, 1\}$  of all the configurations containing at most one 1. We show how some paths can be deformed to the trivial path. It is then easy to show that *all* paths are homotopic to the trivial path. Take an aperture window of size 1, *i.e.*, only one cell is visible at a time. Consider the following path  $p$ , starting at  $(0, (0, 0))$  (we see a 0 at the origin of the  $\mathbb{Z}^2$  plane). The path then moves in the  $\mathbb{Z}^2$  grid while only seeing 0's, and comes back to the origin where it now sees a 1. Then it moves away from the origin while only seeing 0's, and finally comes back to  $(0, 0)$  with a 0 in the window. For simplicity, we also suppose that the path does not pass through the origin at any other time. To sum up, the path is a loop, starting and ending at  $(0, (0, 0))$ , which only sees 0 along the way except at one time ( $t_2$  on the figure) where it sees a 1 at the origin. This is illustrated in Figure 1a.



(a) Example of a path that cannot be traced in a single configuration.

(b) A homotopic deformation to a path that can entirely be traced in the all-0 configuration.

**Figure 1** Example of a path and of a deformation of this path. Notice that the central 0 and 1 windows at  $t_0$  and  $t_2$  are actually located at the same point of the plane, although the figure depicts them on top of each other for the sake of clarity. Red wires can be traced in  $x_0$ , and blue wires in  $x_1$ . The wire of alternating colours can be traced within both, and so it is both homotopic to the initial path, and to the trivial path.

Let  $x_0, x_1$  respectively be the all-zero configuration, and the configuration containing a 1 at the origin. The path  $p$  can be homotopically deformed in the following way: between the times  $t_1$  and  $t_3$ , it can be considered to be entirely in  $x_1$ . It can thus be deformed in this configuration by completely avoiding the origin, and joining the same points, as in Figure 1b. By definition of  $x_1$ , this new path will now see only 0's. The resulting loop then also sees 0's at any point, and so it can be homotopically contracted to the trivial path in the configuration  $x_0$ . This proof can be extended to make any 1 on a path “disappear”, and so any path can be contracted. In this case, this shows that  $\pi_1^{proj}(X, (x_0, (0, 0))) = \{e\}$  is trivial, as the same argument works for arbitrary large  $B_n$ .

## 4 Realization of projective fundamental groups

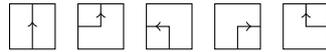
We are now going to prove our main result: any finitely presented group is the fundamental projective group of some SFT.

► **Theorem 1.** *Let  $G = \langle S | R \rangle$  be a finitely presented group. Then, there is a subshift of finite type  $X$  satisfying:*

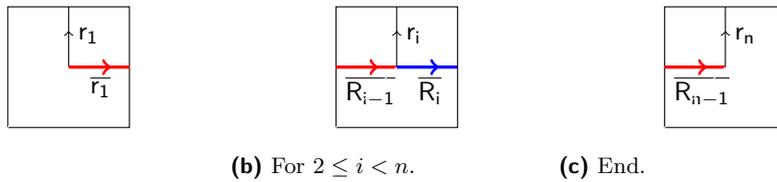
- $X$  is projectively connected,
- the projective fundamental group of  $X$  is isomorphic to  $G$ .

### 4.1 The construction

The subshift  $X$  that we construct will informally consist of oriented wires, drawn on an empty background, each wire corresponding to a generator  $s \in S$  of the group  $G = \langle S | R \rangle$ . We only authorize the wires to go up, perhaps in some kind of “zigzag” manner, but never down or horizontally. More precisely, we define the following tiles: first of all, a tile that we call **empty**, visually represented by  $\square$ , and we denote by  $\mathcal{T}_{\text{empty}}$  the singleton containing this tile. We denote by  $x_{\square} \in X$  the configuration which only contains empty tiles, and its patterns are called **empty patterns**. Then, for each element  $s \in \bar{S} = S \cup \{s^{-1} | s \in S\}$ , we also consider the set  $\mathcal{T}_s$  of the 5 following tiles:



If  $s \neq s'$ , then  $\mathcal{T}_s \cap \mathcal{T}_{s'} = \emptyset$ . Distinct  $\mathcal{T}_s$  will be represented by wires of different colours in the figures. These tiles will, intuitively, be used to represent generators of the group in valid configurations of  $X$ . Finally, we use some other tiles that will play the role of representing the group relations. We can always assume that  $R$  contains the trivial relators  $ss^{-1}$  and  $s^{-1}s$  for all  $s \in S$ . Now, for each relator  $r = r_1 r_2 \dots r_n \in R$ , we let  $\mathcal{T}_r$  be the tiles described by Figure 2.



■ **Figure 2** The relation tiles.

The wire exiting from the right side of the tile Figure 2a does *not* have the same colour as the one exiting from the top. The former colour is denoted by  $\bar{r}_1$ , to differentiate it from the actual  $r_1$  wires. In the other tiles,  $\bar{R}_i = \bar{r}_1 \bar{r}_2 \dots \bar{r}_i$ . Hence, for each relator  $r_1 \dots r_n$ , we have one tile of type Figure 2a and one of type Figure 2c, and  $n - 2$  tiles of type Figure 2b. Tiles belonging to some  $\mathcal{T}_r$  are called relation tiles. Note that if  $u \in R$  is such that it is the prefix of two different relators, *i.e.*, there exists  $v, v' \in \bar{S}^*$  such that  $uv \in R, uv' \in R$  then the colours  $\bar{u}$  are shared by the tiles used to represent those relators and so  $\mathcal{T}_{uv} \cap \mathcal{T}_{uv'} \neq \emptyset$ .  $X$  is the subshift generated by the tileset  $\mathcal{T} = \mathcal{T}_{\text{empty}} \cup \bigcup_{s \in \bar{S}} \mathcal{T}_s \cup \bigcup_{r \in R} \mathcal{T}_r$  along with the obvious adjacency rules: any wire must be extended, by a wire with the same orientation given by the arrows – *e.g.*,  $\begin{bmatrix} \uparrow \\ \uparrow \end{bmatrix}$  and  $\begin{bmatrix} \uparrow \\ \rightarrow \end{bmatrix}$  are forbidden patterns, but  $\begin{bmatrix} \uparrow \\ \rightarrow \\ \rightarrow \end{bmatrix}$  is allowed (assuming the two tiles contain a wire of the same colour).

We now formalize what we really mean by a wire.

► **Definition 13 (Wire).** A **wire** is a sequence  $\mathcal{U} = (T_t, \mathbf{v}_t)_{t \in I}, I \subseteq \mathbb{Z}$  a non-necessarily finite interval, of pairs of non-empty tiles and  $\mathbb{Z}^2$  points, such that

- $\|\mathbf{v}_{t+1} - \mathbf{v}_t\|_1 = 1$ ,
- The tile  $T_{t+1}$  in position  $\mathbf{v}_{t+1}$  extends the wire of tile  $T_t$  in position  $\mathbf{v}_t$ : placing a tile  $\begin{bmatrix} \uparrow \\ \uparrow \end{bmatrix}$  above or below another tile  $\begin{bmatrix} \uparrow \\ \uparrow \end{bmatrix}$  does extend it, while placing it on its right or left side does not, although they are valid patterns of  $X$ .
- $\mathcal{U}$  does not contain two consecutive relation tiles.

► **Remark 14.** We do not prevent a wire from moving back and forth: it is possible to have  $(T_t, \mathbf{v}_t) = (T_{t+2}, \mathbf{v}_{t+2})$ .

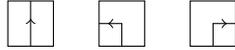
► **Definition 15** (Coherent wire). We say that a wire is *coherent* if there exists a configuration  $x \in X$  such that for any tile  $(T_i, \mathbf{v}_i)$  of the wire,  $x_{\mathbf{v}_i} = T_i$ .

► **Remark 16**. Valid configurations of  $X$  can contain non-intersecting infinite wires, and possibly some relation tiles with wires originating from them. Any relation tile belongs to one horizontal line of  $k$  relation tiles, corresponding to a valid relator  $r_1 \dots r_k$ .

One important concept associated to paths on this subshift is the idea that paths can cross wires. Informally, this is what happens when the window, and in particular, its center, moves from one side to the other of a given wire in a path.

► **Definition 17** (Crossing a wire tile). Let  $n > 0$ , and let  $\mathbf{v}, \mathbf{v}' \in \mathbb{Z}^2$  be two adjacent points, and  $P, P'$  two patterns of respective support  $\mathbf{v} + B_n, \mathbf{v}' + B_n$  such that  $(P, \mathbf{v}), (P', \mathbf{v}')$  is a valid path. For  $(i, j) \in B_n$ , let  $T_{(i,j)}$  be the tile whose bottom-left corner is on  $(i, j)$  in  $P$ . We say that this path crosses a wire tile if

■  $\mathbf{v}' - \mathbf{v} = e_0 = (1, 0)$  (resp.  $-e_0$ ) and the tile  $T_{\mathbf{v}}$  (resp.  $T_{\mathbf{v}-e_0}$ ) was of one of the following forms:



■  $\mathbf{v}' - \mathbf{v} = e_1 = (0, 1)$  (resp.  $-e_1$ ) at the next step  $t + 1$  and the tile  $T_{\mathbf{v}}$  (resp.  $T_{\mathbf{v}-e_1}$ ) was of one of the following form:



In the following, we let  $B_n = \{-n, \dots, n - 1\}^2$ . Unless stated otherwise, all the aperture windows considered will be of this form.

► **Definition 18** (Seeing a wire). A path  $p = (P_i, \mathbf{v}_i)_{i \leq N}$  sees a wire  $\mathcal{U}$  if there exists a timestep  $i \leq N$ , and  $(T_j, \mathbf{v}_j) \in \mathcal{U}$  such that the tile in position  $\mathbf{v}_j$  in  $P_i$  is  $T_j$ .

► **Definition 19** (Crossing a wire). A path crosses a wire if it crosses one of its tiles.

## 4.2 Only Crossed Wires Matter

Our final goal is to prove that the projective fundamental group of this subshift  $X$  is the group  $G = \langle S | R \rangle$ . To do so, the idea will be to associate an element of the group to each path, according to the wires that it crosses. The following lemmas can be seen as a procedure to put paths in some kind of normal form via homotopies, depending only the sequence of crossed wires, regardless of the underlying geometry of the path. All the lemmas consider paths that both start and end in empty patterns, but this is not really a restriction as we will later prove that the subshift  $X$  is projectively connected, and so we will only consider loops based at  $x_{\square}$ . Unless stated otherwise, all the considered paths are using some  $B_n$  as aperture window. We start with some easy statements about patterns of support  $B_n$ , and the wires they may contain.

► **Lemma 20** (Wire Order Lemma). Let  $x \in X$ , and let  $\mathcal{U}, \mathcal{V}$  be two infinite wires in  $x$ . Suppose that  $\mathcal{U}, \mathcal{V}$  do not contain relation tiles.

■ For all  $z \in \mathbb{Z}$ , there exists between one and two  $z_{\mathcal{U}}^0 \in \mathbb{Z}$  such that  $\mathcal{U}$  passes through the position  $(z_{\mathcal{U}}^0, z)$ . If there are two such  $z_{\mathcal{U}}^0$ , then they are necessarily adjacent, e.g.,  $\begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix}$  side-by-side.

■ Let  $z \in \mathbb{Z}$ , and  $z_{\mathcal{U}}^0, z_{\mathcal{V}}^0 \in \mathbb{Z}$  as in the previous point respectively for  $\mathcal{U}$  and  $\mathcal{V}$ . If  $z_{\mathcal{U}}^0 < z_{\mathcal{V}}^0$ , then for all  $z_{\mathcal{U}}, z_{\mathcal{V}}, z \in \mathbb{Z}$  such that  $(z_{\mathcal{U}}, z) \in \mathcal{U}, (z_{\mathcal{V}}, z) \in \mathcal{V}$ , we have  $z_{\mathcal{U}} < z_{\mathcal{V}}$ . Intuitively, this means that wires can globally be ordered from left to right.

If  $\mathcal{U}$  or  $\mathcal{V}$  contains a relation tile, then the previous claims are true only for  $z$  large enough.

► **Remark 21.** Note that the previous lemma is true because we consider wires  $\mathcal{U}, \mathcal{V}$  belonging to some configuration. It is clearly false for arbitrary wires.

► **Lemma 22.** *Let  $P$  be a globally admissible pattern of support  $B_n$  for some  $n > 0$ . Let  $\mathcal{U}$  be a wire in  $P$  without relation tiles. Suppose that  $\mathcal{U}$  passes to the right (resp. left) of  $(0, 0)$  in  $P$ . Then,  $\mathcal{U}$  neither enters nor exits  $P$  on its left (resp. right) edge.*

**Proof.** This directly follows from the fact that no tile contains a horizontal wire, and that  $B_n$  is a square. ◀

► **Corollary 23.** *If  $P$  is a globally admissible pattern that sees a wire  $\mathcal{U}$  with no relation tiles, and  $x \in X$  is such that  $x|_{B_n} = P$ , then  $\sigma_{(0,1)}^{4n}(x)|_{B_n}$  and  $\sigma_{(0,1)}^{-4n}(x)|_{B_n}$  do not see  $\mathcal{U}$ .*

In order to show that the homotopy class of a path  $p$  is indeed only determined by the wires it crosses, we will need several lemmas in which the proof will always be similar: an induction on the length  $L$  of a Coherent path decomposition of  $p$ :

- for  $L = 1$  (i.e.  $p$  is coherent), we explicitly show how to deform  $p$  to obtain the required property.
- for  $L = 2$  we use the Path Co-extensibility Lemma to “normalize” both coherent subpaths of  $p$  using the base case  $L = 1$ .
- In general, if  $p = p_1 * \dots * p_N$ , we can deform both  $p_1$  and  $p_2$  so that  $p \sim p'_1 * p'_2 * \dots * p_N$ , in such a way that we can apply the base case to  $p'_1$ , and the induction case to  $p'_2 * \dots * p_N$ . The key step is therefore to properly show how to deal with the case  $L = 2$ ; this is the purpose of the Path Co-extensibility Lemma that we now show, after some preliminary results.

► **Lemma 24 (Finite Extension Lemma).** *Let  $P$  be an extensible finite pattern of  $X$ , there exists  $x \in X$  containing  $P$ , such that  $x$  contains a finite number of wires.*

► **Definition 25 (Cone).** *For  $n \in \mathbb{N}$ , we define the **cones***

$$C_n^- = \{(i, j) \mid j \leq 0, -|j| - n \leq i < |j| + n\} \quad C_n^+ = \{(i, j) \mid j \geq 0, -j - n \leq i < j + n\}$$

*We denote  $\partial C_n = C_n \cap ((\overline{C_n} + e_0) \cup (\overline{C_n} + e_1))$  the border of a cone.*

► **Lemma 26 (Extensibility Lemma).** *Let  $n > 0$ . There exists  $k > 0$  such that for any  $x \in X$ , there exists  $x' \in X$  with:*

- $x'|_{C_n^\pm} = x|_{C_n^\pm}$
- $x'|_{\sigma_{(0,k)}(C_n^\mp)} = x|_{\sigma_{(0,k)}(C_n^\mp)}$

**Proof.** We prove the case where  $x'$  is empty in a cone above the  $y = 0$  line, and equal to  $x$  below it, the other case being similar. Let  $r$  be the length of the longest relator in the finite presentation of  $G = \langle S|R \rangle$ . Let  $W \subset \mathbb{Z}^2$  be the set of positions of tiles that are part of a wire of  $x$  that:

- either passes by  $C_n^-$
- or originates from a relation tile which is itself part of a relator intersecting  $C_n^-$ .

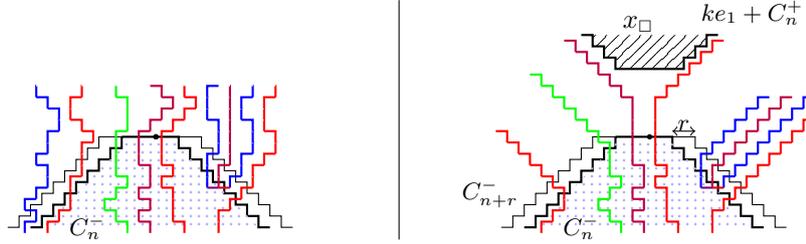
Now, construct  $x'$  as follows:

- for  $(i, j) \in C_{n+r}^- \cap W$ , set  $x'_{(i,j)} = x_{(i,j)}$ . The other tiles of  $C_{n+r}^-$  are empty.
- for  $(i, j) \in \partial C_{n+r}^- \cap W$  and  $j < 0$ , extend the wire above  $(i, j)$  using only tiles  $\boxplus$  and  $\boxminus$  if  $i < 0$ , or  $\boxdot$  and  $\boxdash$  if  $i > 0$ .
- each wire of  $W$  passing by  $(i, 0)$  with  $|i| \leq n + r$  is extended by  $n - |i| + r$  tiles  $\boxplus$ , and then by tiles of the form  $\boxplus$  and  $\boxminus$  if  $i < 0$ , or  $\boxdot$  and  $\boxdash$  if  $i \geq 0$ .
- all the other tiles are empty.

## 75:10 Finitely Presented Groups as Fundamental Groups of Subshifts

Then,  $x'$  is a valid configuration of  $X$  and:

- By definition of  $W$ ,  $x', x$  coincide on  $C_n^-$ .
- $\partial C_{n+r}^-$  contains no relation tile, by definition of  $W$  and  $r$ .
- $(0, n+r+1) + C_n^+$  is empty. See for example Figure 3.



■ **Figure 3** Construction of  $x'$  (on the right) from  $x$  (on the left). In both figures, the central dot is the origin  $(0, 0)$ .

► **Corollary 27** (Path Co-extensibility Lemma). Let  $p = ((P_t, \mathbf{u}_t))_{t \leq N_p}$  and  $q = ((Q_t, \mathbf{v}_t))_{t \leq N_q}$  be two paths with the same aperture window  $B_n$ , satisfying:

- Both  $p$  and  $q$  are coherent paths
- $(P_{N_p}, \mathbf{u}_{N_p}) = (Q_0, \mathbf{v}_0)$  (equivalently,  $p * q$  is well-defined)
- $u_0^1 = v_{N_q}^1$  (i.e.  $q$  ends at the same height as  $p$  starts)

Then, there exists  $p', q', r$  paths such that:

- $r$  ends on an empty pattern
- $p' * r$  and  $r^{-1} * q'$  are well-defined and are both coherent paths.
- $p \sim p'$  and  $q \sim q'$

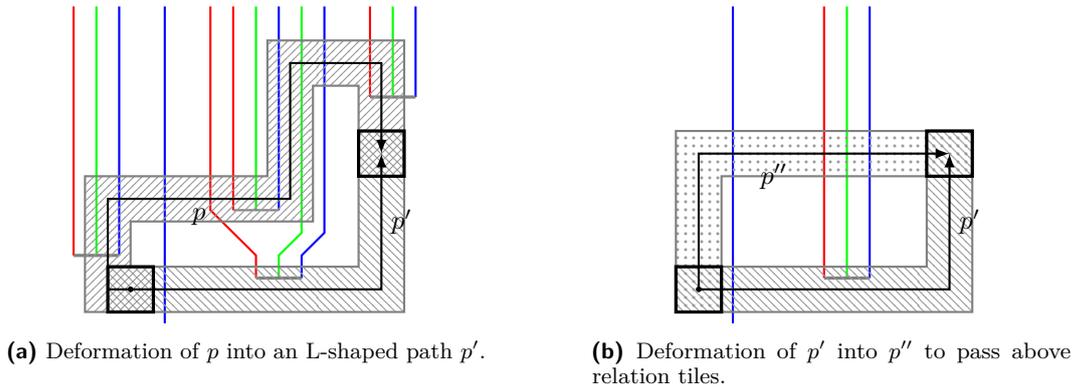
**Proof.** We may assume that  $u_0^1 \leq u_{N_p}^1$ , i.e. the ending point of  $p$  is higher than its starting point, the other case being similar. We can also assume that  $u_{N_p}^1$  is the highest point in the entire trajectory of both  $p$  and  $q$  (we can always homotopically deform  $p$  and  $q$  so that this is true), and up to some shift, we can assume that  $\mathbf{u}_{N_p} = (0, 0)$ . Consider now  $P \subset \mathbb{Z}^2$  so that  $P$  contains all the  $P_t$  and  $Q_t$ . Let  $x_p, x_q$  be configurations in which  $p, q$  can respectively be traced. Take  $N$  large enough so that  $P \subset C_N^-$ . Then, applying the Extensibility Lemma to  $x_p, N$  on one hand,  $x_q, N$  on the other hand, gives two configurations  $x'_p, x'_q \in X$ . Let  $r$  be the path obtained by moving up for  $2N + 1$  steps in either  $x'_p$  or  $x'_q$ , starting from the origin, which is the same path in both cases. Then  $r$  satisfies the conditions of Path Co-extensibility Lemma.

We are now ready to prove the main lemmas needed to show Theorem 1.

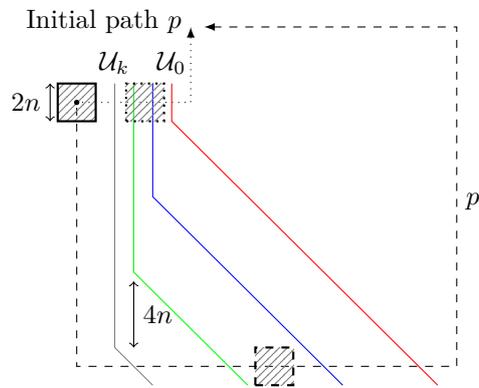
► **Lemma 28** (No Relation Tile Lemma). Let  $p$  be a path starting and ending on an empty pattern. Then there exists  $p' \sim p$  that does not contain any relation tile.

**Proof.** As explained above, the proof is by induction on the length of a coherent path decomposition of  $p$ . The base case when  $p$  is a coherent path is illustrated in Figure 4. See the appendix of the full version for the full proof.

► **Lemma 29** (Single Wire Lemma). Let  $p = (P_i, \mathbf{v}_i)_{0 \leq i \leq N}$  be a path starting and ending with empty patterns. There exists a path  $p'$ , homotopic to  $p$ , such that the union of any two consecutive patterns in  $p'$  contains at most a single wire.



■ **Figure 4** A coherent path deformed so as not to see relation tiles.



■ **Figure 5** Deformation of  $p$  into  $p'$  in a single configuration to see only one wire per pattern.

**Proof.** As for the No Relation Tile Lemma, we illustrate in Figure 5 the case where  $p$  is itself coherent. For the full proof, see the appendix of the full version. ◀

► **Lemma 30 (No Uncrossed Wire Lemma).** *Let  $p$  be a path starting and ending with empty patterns, and  $\mathcal{U}$  some wire seen but not crossed by  $p$ . There exists a path  $p'$ , homotopic to  $p$ , which does not see  $\mathcal{U}$ .*

**Proof.** The idea is that using the previous Single Wire Lemma, we can deal with each wire independently. In particular, the uncrossed wire  $\mathcal{U}$  is the only wire seen by some subpath  $p'$  of  $p$ , and is not seen by  $p$  neither before nor after  $p'$ . Hence, it suffices to show the result for paths seeing a single wire overall. In that case, one observes that  $\mathcal{U}$  has to stay in the same “side” of the aperture window along  $p'$ , that can therefore be deformed without crossing  $\mathcal{U}$  by moving sufficiently far in the opposite direction. For more details, see the appendix of the full version. ◀

► **Lemma 31 (Cross Anywhere Lemma).** *Let  $p$  be a path starting and ending with empty patterns. If  $p$  sees no relation tiles, but sees and crosses a single wire  $\mathcal{U}$  exactly once, then for all  $\mathbf{v} = (v^0, v^1) \in \mathbb{Z}^2$ ,  $p$  is homotopic to a path  $p'$  which crosses  $\mathcal{U}$  exactly on  $\mathbf{v}$ .*

**Proof.** The idea is that if  $\mathcal{U}$  exits the aperture window  $B_n$  of  $p$  in position  $(i, j) \in \mathbb{Z}^2$ , it can be extended using tiles  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$  and  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$ , or  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$  and  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$ , to pass anywhere inside  $(i, j) + C_n^-$  or  $(i, j) + C_n^+$ . The path  $p$  can then be deformed to cross it anywhere in those two cones. Using

several such deformations, we can deform  $p$  so that it crossed  $\mathcal{U}$  anywhere in the plane. Note that even if  $p$  is initially coherent, it might happen that  $p'$  is not, depending on  $\mathbf{v}$  and where  $p$  initially crossed  $\mathcal{U}$ . See the appendix of the full version for the complete proof. ◀

### 4.3 Projective connectedness

► **Lemma 32** (Projective connectedness).  $X$  is projectively connected.

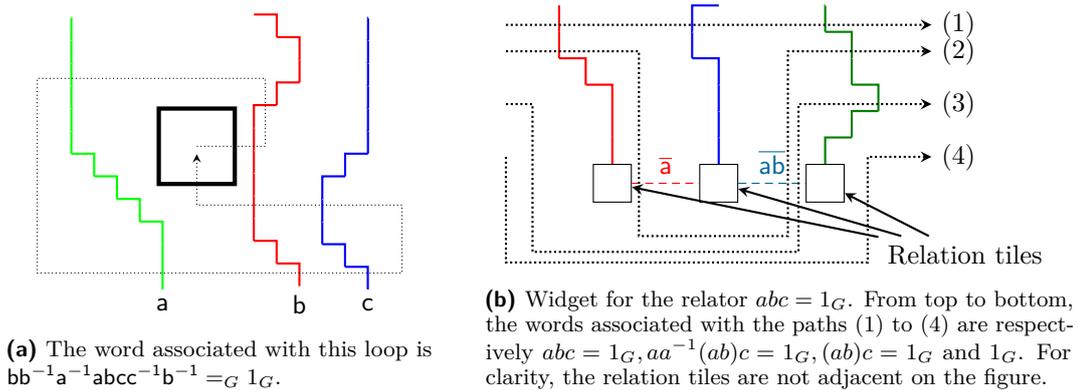
**Proof.** The proof relies on the Extensibility Lemma. The idea is that starting from any configuration  $x$ , there always exists a configuration  $x'$  containing a infinite cone (see Definition 25) of  $x_\square$ , and an infinite cone of  $x$ . We can then use this configuration to construct for  $n > 0$  a path  $p_n$  with aperture window  $B_n$  that first moves sufficiently far into the latter cone in  $x$ , then to the former cone in the configuration  $x'$ , and finally comes back to the origin in  $x_\square$ . See the appendix of the full version for the precise proof. ◀

### 4.4 Computing the projective fundamental group

We can now compute  $\pi_1^{proj}(X)$ , which is independent of the basepoint since  $X$  is projectively connected. Hence, unless stated otherwise, all the loops in this proof are based at  $(x_\square, (0, 0))$ . With any such loop  $p$ , we associate a word  $\llbracket p \rrbracket$  on the alphabet  $\bar{S}$  in the following way:

- If  $p$  does not cross any wire, we associate the empty word with it,  $\llbracket p \rrbracket = \varepsilon$ .
- If  $p$  crosses a single wire  $\mathcal{U}$ , then:
  - If  $\mathcal{U}$  is not a horizontal wire found on a relation tile, and  $s \in \bar{S}$  is the generator corresponding to  $\mathcal{U}$  (see Subsection 4.1)
    - \* if  $p$  crosses it from left to right, or from top to bottom on a tile shaped as  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$ , or from bottom to top on a tile  $\begin{smallmatrix} \square \\ \square \end{smallmatrix}$ , then  $\llbracket p \rrbracket = s \in \bar{S}$ .
    - \* if  $p$  crosses it in any other direction, we set  $\llbracket p \rrbracket = s^{-1} \in \bar{S}$
  - Otherwise,  $\mathcal{U}$  is a horizontal wire on a relation tile. Let  $\bar{R}_i = \bar{r}_0 \dots \bar{r}_i$  be its colour.
    - \* If it is crossed from top to bottom, then  $\llbracket p \rrbracket = \bar{r}_i^{-1} \dots \bar{r}_0^{-1} \in \bar{S}^*$
    - \* Otherwise,  $\llbracket p \rrbracket = R_i = r_0 \dots r_i$
- If  $p = p_1 * p_2$ , then  $\llbracket p \rrbracket = \llbracket p_1 \rrbracket \cdot \llbracket p_2 \rrbracket \in \bar{S}^*$  where  $\cdot$  represents the concatenation in  $\bar{S}^*$ .

Some examples are given in Figure 6a and Figure 6b.



■ **Figure 6** Some examples of words and group elements associated with coherent paths.

For any two words  $w, w'$  on  $\bar{S}$ , we write  $w \equiv w'$  if they are equal as words on this alphabet, and  $w =_G w'$  if they represent the same element of the group  $G$ . Let  $\leftrightarrow_R$  be the relation defined as the symmetric closure of  $\{(uvw, uv) \mid w \in R \text{ and } u, v \in (\bar{S})^*\}$ , corresponding to the operation of inserting and removing relators to words. We can always suppose that it is

reflexive by adding the empty word  $\varepsilon$  to the relators. We denote  $\leftrightarrow_R^*$  its transitive closure. By definition,  $w \leftrightarrow_R^* w' \iff w =_G w'$  (see e.g., [28, Theorem 1.1]). For example, if we take  $a \in S$ , we have  $aa^{-1} =_G 1_G$ , but  $aa^{-1} \neq \varepsilon$ .

In order to prove that the projective fundamental group of this subshift is  $G$ , we will prove that the operation  $\llbracket p \rrbracket$  entirely characterizes a loop up to homotopy, in the sense that loops associated with the same element of  $G$  are exactly a projective loop-class:

► **Lemma 33** (Homotopic Implies Equal). *For any window  $B_n$  and for any pair of loops  $p_n, p'_n$  starting at  $(x_{\square|B_n}, (0, 0))$ ,  $p_n \sim_{B_n} p'_n \implies \llbracket p_n \rrbracket =_G \llbracket p'_n \rrbracket$ .*

► **Lemma 34** (Equal Implies Homotopic). *For any window  $B_n$  and for any pair of loops  $p_n, p'_n$  starting at  $(x_{\square|B_n}, (0, 0))$ ,  $\llbracket p_n \rrbracket =_G \llbracket p'_n \rrbracket \implies p_n \sim_{B_n} p'_n$ .*

The full proofs can be found in the appendix of the full version.

► **Theorem 35.**  $\pi_1^{proj}(X) = G$

**Proof.** Let  $n > 0$  and let  $\Phi_n: p \in \pi_1^{B_n}(X, (x_{\square}, (0, 0))) \mapsto \llbracket p \rrbracket \in G$  be the function which associates with a loop-class with aperture window  $B_n$  the corresponding element of  $G$ . The Homotopic Implies Equal and Equal Implies Homotopic show that it is well-defined and injective. Let  $[p], [p']$  be two loop-classes based at  $(x_{\square|B_n}, (0, 0))$ . We have shown that  $[p] \sim_{B_n} [p'] \iff \Phi_n([p]) =_G \Phi_n([p'])$ . Now notice that  $\Phi_n([p * p']) =_G \Phi_n(p) \cdot_G \Phi_n(p')$ , i.e.,  $\Phi_n$  is a group morphism. To show that it is surjective, let  $g \in G$  any element, and  $u_1 \dots u_n \in \bar{S}^*$  such that  $u_1 \dots u_n =_G g$ . Let  $x^g$  the following configuration:

- For  $1 \leq i \leq \ell$  and  $j \in \mathbb{Z}$ ,  $x^g(i, j)$  is a tile of type  $\square$  and of colour  $u_i$
- Otherwise,  $x^g(i, j) = \square$

Now, consider the following loop: define  $p_n$  as the loop based at  $(x_{\square|B_n}, (0, 0))$ , which:

- moves left for  $n$  steps in  $x_{\square}$
- moves right for  $2n + \ell$  steps in  $x^g$  – at this point, it sees an empty pattern, after having crossed all the wires of  $x^g$
- comes back to  $(0, 0)$  in  $x_{\square}$ .

By definition,  $\llbracket p_n \rrbracket \equiv u_1 \dots u_n =_G g$ .

Furthermore, notice that for any loop-class  $[p_{n+1}]$  based at  $(x_{\square|B_{n+1}}, (0, 0))$ , if  $p_{n+1}$  projects down to  $p$  then  $\Phi_{n+1}([p_{n+1}]) =_G \Phi_n([p])$ . This shows that  $\pi_1^{proj}(X, (x_{\square}, (0, 0)))$  is isomorphic to  $G$ , and the final result follows from the fact that  $X$  is projectively connected. ◀

---

## References

- 1 Nathalie Aubrun, Sebastián Barbieri, and Emmanuel Jeandel. About the domino problem for subshifts on groups. In *Trends in Mathematics*, pages 331–389. Springer International Publishing, 2018. doi:10.1007/978-3-319-69152-7\_9.
- 2 Nathalie Aubrun, Sebastián Barbieri, and Mathieu Sablik. A notion of effectiveness for subshifts on finitely generated groups. *Theoretical Computer Science*, 661:35–55, January 2017. doi:10.1016/j.tcs.2016.11.033.
- 3 Nathalie Aubrun and Jarkko Kari. Tiling Problems on Baumslag-Solitar groups. In *Machines, Computations and Universality (MCU)*, number 128 in Electronic Proceedings in Theoretical Computer Science, pages 35–46, 2013. doi:10.4204/EPTCS.128.12.
- 4 Nathalie Aubrun and Mathieu Sablik. An order on sets of tilings corresponding to an order on languages. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009*, pages 99–110, 2009.
- 5 Alexis Ballier, Bruno Durand, and Emmanuel Jeandel. Tilings robust to errors. In *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, pages 480–491, 2010. doi:10.1007/978-3-642-12200-2\_42.

- 6 Robert Berger. *The Undecidability of the Domino Problem*. Number 66 in Memoirs of the American Mathematical Society. The American Mathematical Society, 1966.
- 7 William W. Boone. Certain Simple, Unsolvable Problems of Group Theory V. *Indagationes Mathematicae*, 60:22–27, 1957. doi:10.1016/S1385-7258(57)50003-6.
- 8 William W. Boone. Word problems and recursively enumerable degrees of unsolvability. a sequel on finitely presented groups. *The Annals of Mathematics*, 84(1):49, July 1966. doi:10.2307/1970530.
- 9 Mike Boyle, Ronnie Pavlov, and Michael Schraudner. Multidimensional sofic shifts without separation and their factors. *Transactions of the AMS*, 362(9):4617–4653, September 2010. doi:10.1090/S0002-9947-10-05003-8.
- 10 Nishant Chandgotia and Brian Marcus. Mixing properties for hom-shifts and the distance between walks on associated graphs. *Pacific Journal of Mathematics*, 294(1):41–69, January 2018. doi:10.2140/pjm.2018.294.41.
- 11 John H. Conway and Jeffrey C. Lagarias. Tiling with polyominoes and combinatorial group theory. *J. Comb. Theory, Ser. A*, 53(2):183–208, 1990. doi:10.1016/0097-3165(90)90057-4.
- 12 Van Cyr and Bryna Kra. The automorphism group of a shift of linear growth: beyond transitivity. *Forum of Mathematics, Sigma*, 3, February 2015. doi:10.1017/fms.2015.3.
- 13 Sebastián Donoso, Fabien Durand, Alejandro Maass, and Samuel Petite. On automorphism groups of low complexity subshifts. *Ergodic Theory and Dynamical Systems*, 36(1):64–95, November 2015. doi:10.1017/etds.2015.70.
- 14 Bruno Durand, Andrei Romashchenko, and Alexander Shen. Effective Closed Subshifts in 1D Can Be Implemented in 2D. In *Fields of Logic and Computation*, number 6300 in Lecture Notes in Computer Science, pages 208–226. Springer, 2010. doi:10.1007/978-3-642-15025-8\_12.
- 15 Francesca Fiorenzi. Periodic configurations of subshifts on groups. *International Journal of Algebra and Computation*, 19(03):315–335, May 2009. doi:10.1142/s0218196709005123.
- 16 Silvere Gangloff, Benjamin Hellouin de Menibus, and Piotr Oprocha. Short-range and long-range order: a transition in block-gluing behavior in hom shifts, 2022. doi:10.48550/arXiv.2211.04075.
- 17 William Geller and James Propp. The projective fundamental group of a  $\mathbb{Z}^2$ -shift. *Ergodic Theory and Dynamical Systems*, 15(6):1091–1118, 1995.
- 18 Pierre Guillon, Emmanuel Jeandel, Jarkko Kari, and Pascal Vanier. Undecidable word problem in subshift automorphism groups. In *Computer Science – Theory and Applications*, pages 180–190. Springer International Publishing, 2019. doi:10.1007/978-3-030-19955-5\_16.
- 19 Yuri Gurevich and I Koryakov. Remarks on Berger’s paper on the domino problem. *Siberian Math. Journal*, pages 319–320, 1972.
- 20 David Harel. Recurring Dominoes: Making the Highly Undecidable Highly Understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.
- 21 Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.
- 22 G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical Systems Theory*, 3(4):320–375, December 1969. doi:10.1007/bf01691062.
- 23 Michael Hochman. On the dynamics and recursive properties of multidimensional symbolic systems. *Inventiones Mathematicae*, 176(1):2009, April 2009.
- 24 Michael Hochman. On the automorphism groups of multidimensional shifts of finite type. *Ergodic Theory and Dynamical Systems*, 30(3):809–840, 2010. doi:10.1017/S0143385709000248.
- 25 Michael Hochman and Tom Meyerovitch. A characterization of the entropies of multidimensional shifts of finite type. *Annals of Mathematics*, 171(3):2011–2038, May 2010. doi:10.4007/annals.2010.171.2011.
- 26 Emmanuel Jeandel. Translation-like Actions and Aperiodic Subshifts on Groups. working paper or preprint, August 2015. URL: <https://hal.inria.fr/hal-01187069>.
- 27 Emmanuel Jeandel and Pascal Vanier. A characterization of subshifts with computable language. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPICs*, pages 40:1–40:16. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.40.

- 28 Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory: Presentations of groups in terms of generators and relations*. Courier Corporation, 2004.
- 29 Kengo Matsumoto. Dimension groups for subshifts and simplicity of the associated  $c^*$ -algebras. *Journal of the Mathematical Society of Japan*, 51(3), July 1999. doi:10.2969/jmsj/05130679.
- 30 Tom Meyerovitch. Growth-type invariants for  $\mathbb{Z}^d$  subshifts of finite type and arithmetical classes of real numbers. *Inventiones Mathematicae*, 184(3), 2010. doi:10.1007/s00222-010-0296-1.
- 31 P.S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, 44:3–143, 1955.
- 32 Ronnie Pavlov and Michael Schraudner. Entropies realizable by block gluing  $\mathbb{Z}^d$  shifts of finite type. *Journal d'Analyse Mathématique*, 126(1):113–174, April 2015. doi:10.1007/s11854-015-0014-4.
- 33 Marcus Pivato. Algebraic invariants for crystallographic defects in cellular automata. *Ergodic Theory and Dynamical Systems*, 27(1):199–240, 2007.
- 34 Klaus Schmidt. The cohomology of higher-dimensional shifts of finite type. *Pacific Journal of Mathematics*, 170(1):237–269, 1995.
- 35 Klaus Schmidt. Tilings, fundamental cocycles and fundamental groups of symbolic  $\mathbb{Z}^d$ -actions. *Ergodic Theory and Dynamical Systems*, 18(6):1473–1526, 1998.
- 36 William P Thurston. Conway's tiling groups. *The American Mathematical Monthly*, 97(8):757–773, 1990.
- 37 Marcin Wrochna. Homomorphism reconfiguration via homotopy. *SIAM Journal on Discrete Mathematics*, 34(1):328–350, January 2020. doi:10.1137/17m1122578.



# Deciding Predicate Logical Theories Of Real-Valued Functions

Stefan Ratschan   

Institute of Computer Science of the Czech Academy of Sciences, Prague, Czech Republic

---

## Abstract

The notion of a real-valued function is central to mathematics, computer science, and many other scientific fields. Despite this importance, there are hardly any positive results on decision procedures for predicate logical theories that reason about real-valued functions. This paper defines a first-order predicate language for reasoning about multi-dimensional smooth real-valued functions and their derivatives, and demonstrates that – despite the obvious undecidability barriers – certain positive decidability results for such a language are indeed possible.

**2012 ACM Subject Classification** Mathematics of computing → Continuous functions; Theory of computation → Automated reasoning; Theory of computation → Logic and verification

**Keywords and phrases** decision procedures, first-order predicate logical theories, real numbers, real-valued functions

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.76

**Related Version** *Full Version:* <https://arxiv.org/abs/2306.16505>

**Funding** *Stefan Ratschan:* This work was supported by the project 21-09458S of the Czech Science Foundation GA ČR and institutional support RVO:67985807.

## 1 Introduction

Predicate logical decision procedures have become a major workhorse in computer science, for example, as the basic reasoning engines in SAT modulo theory (SMT) solvers [3]. Common decision procedures support theories such as uninterpreted function symbols, arrays, linear integer arithmetic, and real arithmetic. However, many areas of computer science (e.g., computer aided design, formal verification of physical systems, machine learning) use as their basic data structure not only real numbers but real-valued *functions*, for example, to represent solid objects [13], correctness certificates [30, 29] or neural networks [1]. Moreover, due to their fundamental role as a basic mathematical object, real-valued functions are used as a basic modeling tool throughout many further scientific areas. But unfortunately, real-valued functions have been left almost completely untouched by research on predicate logical decision procedures. The goal of this paper is to take a first step to fill this gap.

More concretely, the paper provides the following contributions:

- We formalize a first-order language of real-valued functions that allows reasoning about both real numbers and multi-dimensional real-valued smooth functions based on the usual arithmetical operations, function evaluation and differentiation.
- We prove that a quantifier-free fragment of the language that restricts arithmetic to addition and multiplication of real numbers, but still provides function evaluation and differentiation, is decidable.
- We prove that for a fragment of the language that keeps the restriction of arithmetic to addition and multiplication of real numbers but allows arbitrary quantification on real-valued variables (but not on function-valued variables), there is an algorithm that can detect satisfiability for all input formulas that are robustly satisfiable in the sense that there is a satisfying assignment that stays satisfying under small perturbations of the values of function-valued variables.



© Stefan Ratschan;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 76; pp. 76:1–76:15

Leibniz International Proceedings in Informatics



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

We neither claim theoretical nor practical efficiency of the resulting decision procedures. Instead, our goal is to overcome scientific fragmentation by developing a framework that can be instantiated to more efficient techniques for specific applications.

The paper has the following structure: In the next section, we discuss related work. In Section 3, we define the syntax and semantics of the mentioned predicate language for reasoning about smooth real-valued functions. In Section 4 we prove decidability of the quantifier-free case. In Section 5 we discuss decidability of the case with arbitrary quantification on real-valued variables. In Section 6 we conclude the paper.

## 2 Related Work

Reasoning about real-valued functions – that we also simply call *real functions* – will, of course, be usually based on reasoning about real numbers. This is facilitated by the fact that unlike the case of the integers, in the case of the real numbers, its non-linear theory (i.e., the theory of real closed fields) is decidable [38]. The decidability of the case with the exponential function is still unknown, but is decidable provided Schanuel’s conjecture holds [25]. Inclusion of the sine function makes the problem undecidable since – as a periodic function – it is able to encode the integers. This makes any theory that allows reasoning about systems of linear ordinary differential equations (ODEs) undecidable, since the sine function appears as the solution of the linear ODE  $\dot{x} = -y, \dot{y} = x$ .

In mathematical analysis, real functions are often abstracted to elements of abstract function spaces such as Banach spaces and Hilbert spaces [24]. However, with one notable exception [37] we are aware of, corresponding predicate logical decision problems have been largely ignored by computer science.

An important occurrence of real functions is in the role of solutions of ordinary differential equations (ODEs) and hybrid dynamical systems. Formal verification of such systems has been an active research topic over many years [10], with a plethora of decidability and undecidability results [16, 8, 23, 4, 5]. Deductive verification bases formal verification on automated reasoning frameworks such as hybrid dynamic logic [29], or proof assistants such as Isabelle/HOL [15]. Reasoning with functions as the solution of ODEs has been included into SAT solvers without formulation as a first-order decision problem [12, 18]. ODEs have also played a role as objects in constraint programming [21]. In contrast to the work mentioned in this paragraph, in this paper, we introduce a general logical language with variables and predicate and function symbols ranging over real-valued functions. Especially, we allow multi-dimensional functions and partial differentiation, whereas ODEs and hybrid systems are defined using one-dimensional functions, only (the single dimension being time).

Computation in function spaces plays a major role in numerical analysis, where it is mostly restricted to representing solutions to certain specific computation problems, especially, solving ordinary or partial differential equations. There are also some general approaches to computing with functions [11, 9]. However, the basic assumption in numerical analysis is that the solution to the given problem exists and is unique, and the goal is to compute an approximation of this solution, whereas in this paper we consider satisfiability questions, where a proof of existence is the goal, not an assumption.

Computer algebra [41] studies computation with symbolic objects, especially polynomials, that can be interpreted as representations of real functions. Unlike that, in this paper we are interested in solving problems of reasoning about functions that are independent from a certain representation.

The proof of decidability of the quantifier-free case will be based on abstracting function variables to uninterpreted function symbols. Abstraction to uninterpreted function symbols is a classical technique in formal verification [6] that has also been applied to real functions [7], but with the goal of modeling *specific* function symbols, while in this paper we are interested in general reasoning about smooth real functions and their derivatives.

For quite some time, robustness has been recognized as tool for characterizing solvable cases of undecidable decision problems. It was used for dynamical systems [16, 2] and for decision procedures for the real numbers [32, 17]. However, all of those results do not allow a general language for reasoning about real functions.

### 3 Formal Syntax and Semantics

In this section, we define the syntax and semantics of the first-order language for reasoning about real functions that we will want to decide. As a first example, consider the formula

$$\exists X \forall u, v . \text{app}(\partial_1 X, u, v) = 1 \wedge \text{app}(\partial_2 X, u, v) \leq u^2,$$

that asks the question whether there exists a smooth function in  $\mathbb{R}^2 \rightarrow \mathbb{R}$  whose partial derivative in its first argument is one everywhere, and whose partial derivative in its second argument is less or equal the square of its first argument. The reader will find more examples at the beginning of each of the two following sections.

The language will be sorted, allowing variables that range over real numbers and variables that range over real functions. We denote the sort ranging over real numbers, that we also call the *scalar sort*, by  $\mathcal{R}$ , and the sorts ranging over real-valued functions, that we also call the *function sorts*, by  $\mathcal{F}_n$ , where  $n \in \mathbb{N}$  refers to the number of arguments (i.e., dimension of the domain). We will also use the symbol  $\mathcal{F}$  to stand for any sort  $\mathcal{F}_i, i \in \mathbb{N}$ . For each of those sorts, we assume a corresponding set of variables  $\mathcal{V} = \mathcal{V}_{\mathcal{R}} \cup \mathcal{V}_{\mathcal{F}_1} \cup \mathcal{V}_{\mathcal{F}_2} \dots$ . We will write the elements of  $\mathcal{V}_{\mathcal{R}}$  using lower-case letters and the elements of  $\mathcal{V}_{\mathcal{F}_1} \cup \mathcal{V}_{\mathcal{F}_2} \dots$  using upper case letters. We will also use the symbol  $\mathcal{V}_{\mathcal{F}}$  to denote the set of all function variables  $\mathcal{V}_{\mathcal{F}_1} \cup \mathcal{V}_{\mathcal{F}_2} \dots$ .

We will build formulas based on the usual syntax of many-sorted first-order logic. Here, we allow rational constants, arithmetical function symbols such as  $+$ ,  $\times$ ,  $\exp$ ,  $\sin$ , and predicate symbols  $=, \leq, \geq, <, >$  that have the usual arity. For every  $n \in \mathbb{N}$ , we allow the function symbols  $\text{app} : \mathcal{F}_n \times \mathcal{R}^n \rightarrow \mathcal{R}$  and  $\partial_i : \mathcal{F}_n \rightarrow \mathcal{F}_n, i \in \{1, \dots, n\}$  that we call app-operator and differentiation operator, respectively. As usual, we will often write the differentiation operator without parenthesis, and for  $X \in \mathcal{F}_1$ , we also write  $\dot{X}$  instead of  $\partial_1 X$ . We will also call a term whose outermost symbol is the function symbol  $\text{app}$ , an *app-term*.

We will call formulas whose function symbols are restricted to  $\{+, \times, \text{app}\} \cup \{\partial_i \mid i \in \mathbb{N}\}$ , and hence avoiding transcendental function symbols, *function-algebraic*. As already mentioned in the introduction, in this paper, we concentrate on this case.

We define the semantics of formulas by defining a structure  $\mathfrak{A}$  giving the usual real-valued semantics to all function and predicate symbols. This allows us to avoid questions of axiomatization and, at the same time, ensures compatibility with the common intuition. Clearly, satisfiability of a formula based on classical mathematical semantics, implies its satisfiability wrt. an arbitrary axiomatization compatible with classical mathematics.

In more detail, the structure  $\mathfrak{A}$  will be many-sorted, where the sort  $\mathcal{R}$  ranges over the real numbers  $\mathbb{R}$  and the sorts  $\mathcal{F}_n, n \in \mathbb{N}$  range over the set of smooth (i.e., infinitely often differentiable) functions in  $\mathbb{R}^n \rightarrow \mathbb{R}$ . We will use the notation that for any smooth function  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ , and tuple  $(\beta_1, \dots, \beta_n) \in \mathbb{N}_0^n$ ,  $D^{(\beta_1, \dots, \beta_n)} F$  denotes the function that is the result

of differentiating for every  $i \in \{1, \dots, n\}$  the function  $F$   $\beta_i$ -times wrt. its  $i$ -th component. The semantics of function and predicate symbols on the real numbers will be as usual. The app-operator and differentiation operator are defined as follows:

- For every  $n \in \mathbb{N}$ , for all  $X \in \mathcal{F}_n$ ,  $\mathfrak{A}(\text{app})(X, x_1, \dots, x_n) = X(x_1, \dots, x_n)$  (i.e., function application in its usual mathematical sense)
- For every  $n \in \mathbb{N}$ ,  $i \in \{1, \dots, n\}$ , for every  $X \in \mathcal{F}_n$ ,  $\mathfrak{A}(\partial_i)(X) = D^d X$ , where  $d \in \mathbb{N}_0^n$  with  $d(i) = 1$  and for every  $k \neq i$ ,  $d(k) = 0$  (i.e., the result of taking the derivative of  $X$  wrt. its  $i$ -th argument).

We will denote the set of variable assignments assigning to each variable an element of its respective domain, by  $\aleph$ . Given an assignment  $\alpha \in \aleph$  we can now assign semantics to formulas in the usual way, writing  $\alpha \models \phi$  iff the interpretation given by structure  $\mathfrak{A}$  and assignment  $\alpha$  satisfies  $\phi$ . We call a formula  $\phi$  *satisfiable* iff there is an assignment  $\alpha \in \aleph$  such that  $\alpha \models \phi$ . In such a case we will also say that  $\phi$  is  $\mathcal{F}$ -satisfiable. By abuse of notation, we will use the symbol  $\mathcal{F}$  to not only denote the function sorts, but also the *theory*  $\mathcal{F}$  of  $\mathcal{F}$ -satisfiable formulas.

#### 4 Quantifier-Free Case

In this section, we consider formulas that are quantifier-free and function-algebraic. Here are some examples:

- $\text{app}(X, t) \geq 1 \wedge \text{app}(X, t+1)^2 \leq 1$ : This formula restricts the value of the function  $X$  at two different points  $t$  and  $t+1$ . Since these points are different, for checking satisfiability of the formula, it suffices check satisfiability of the algebraic inequalities  $r \geq 1 \wedge s^2 \leq 1$ . Based on a satisfying assignment for this formula, we get a satisfying assignment for the original formula by assigning to  $X$  a function interpolating between the values for  $r$  and  $s$ .
- $\text{app}(X, 0) = 0 \wedge \text{app}(\dot{X}, 1) = \text{app}(X, 1)^2$ . This formula not only restricts values for the function  $X$ , but also states a relationship between the value of  $X$  and its derivative. The formula is satisfiable since the identity function satisfies the properties stated by the formula.
- $\text{app}(\partial_1 X, t) = 1 \wedge \text{app}(\partial_2 X, t) = 1$ . This formula states a relationship between two partial derivatives of  $X$  at the same value  $t$ . This holds, for example, for the function  $X$  with  $X(u, v) = u + v$ .

The basic idea for deciding such formulas is, that quantifier-free formulas constrain the values of function variables only at a finite (but not fixed) subset of their domain which will allow us to treat them as uninterpreted function symbols. To do so, we have to get rid of the app- and differentiation operators. For this, observe that the only syntactic elements that result in terms of function sort are function variables and differentiation operators. Hence, differentiation operators can only occur in the form of terms of the form  $\partial_{i_1} \partial_{i_2} \dots \partial_{i_n} V$ . So we let  $\tau_\partial(\phi)$  be the formula resulting from replacing every maximal term of this form (i.e., every term of this form that is not an argument of a differentiation operator) by a fresh function variable  $V_{i_1, \dots, i_n}$ . For example,

$$\tau_\partial(\text{app}(\partial_1 X, t) + \text{app}(\partial_2 X, t) = 1) \equiv \text{app}(X_1, t) + \text{app}(X_2, t) = 1.$$

The next step is to get rid of the app operator. For this, we denote, for every quantifier-free formula  $\phi$ , by  $\tau(\phi)$  the result of replacing every app-term  $\text{app}(X, t_1, \dots, t_k)$  of  $\tau_\partial(\phi)$  by  $X(t_1, \dots, t_k)$  where in the resulting formula, we now consider  $X$  a  $k$ -ary function symbol.

Continuing the example, we get

$$\tau(\text{app}(\partial_1 X, t) + \text{app}(\partial_2 X, t) = 1) \equiv X_1(t) + X_2(t) = 1.$$

The resulting formula is a formula in the language defined by the combination of the signature of the theory of real-closed fields and the signature of the theory of uninterpreted function symbols. The combination of these two theories, that we will denote by  $\mathcal{RU}$ , is decidable: The signatures of the theory of uninterpreted function symbols and the theory of real-closed fields only share equality, and both are stably infinite<sup>1</sup>, hence the decision procedures for the individual procedures can be combined to a decision procedure for the combined theory by the Nelson-Oppen theory combination procedure [27, 6]. As a consequence, we can algorithmically decide  $\mathcal{RU}$ -satisfiability of translated formulas  $\tau(\phi)$ . Moreover, the translation preserves satisfiability:

► **Theorem 1.** *A conjunctive formula  $\phi$  is  $\mathcal{F}$ -satisfiable if and only if  $\tau(\phi)$  is  $\mathcal{RU}$ -satisfiable.*

For proving this theorem, we have to bridge two differences between  $\mathcal{F}$ - and  $\mathcal{RU}$ -satisfiability: First, the semantics of  $\mathcal{F}$ -satisfiability restricts the domain of function variables to specific functions, more concretely, to smooth real function. And second, the theories of real closed fields and uninterpreted function symbols are defined using axioms, unlike our theory  $\mathcal{F}$  that we defined semantically, by fixing a certain structure. Before going into the details of the proof, we state a few lemmas. The first one extracts the non-algorithmic core of the Nelson-Oppen method [27, 39, 6]:

► **Lemma 2.** *Let  $T_1$  and  $T_2$  be two stably infinite theories of respective signatures  $\Sigma_1$  and  $\Sigma_2$ , having only equality in common. Let  $\phi_1$  be a conjunctive  $\Sigma_1$ -formula, and  $\phi_2$  a conjunctive  $\Sigma_2$ -formula. Then  $\phi_1 \wedge \phi_2$  is  $(T_1 \cup T_2)$ -satisfiable iff there is an equivalence relation  $E$  on the common variables  $V := \text{var}(\phi_1) \cap \text{var}(\phi_2)$  s.t.  $\phi_1 \wedge \rho(V, E)$  is  $T_1$ -satisfiable and  $\rho(V, E) \wedge \phi_2$  is  $T_2$ -satisfiable, where  $\rho(V, E)$  is the formula*

$$\bigwedge_{u,v \in V . uEv} u = v \wedge \bigwedge_{u,v \in V . \neg(uEv)} u \neq v.$$

Every  $(\Sigma_1 \cup \Sigma_2)$ -formula  $\phi$  can be brought into an equi-satisfiable formula of the form  $\phi_1 \wedge \phi_2$ , where  $\phi_1$  is a  $\Sigma_1$ -formula, and  $\phi_2$  is a  $\Sigma_2$ -formula using the so-called variable abstraction phase of the Nelson-Oppen method. In our case,  $T_1$  is the theory of real closed fields, and  $T_2$  the theory of uninterpreted function symbols. For the result

$$X(t) \geq 1 \wedge X(t+1)^2 \leq 1,$$

of translating the first example from the beginning of the section, the result of the variable abstraction phase is the equi-satisfiable formula

$$v_1 \geq 1 \wedge v_2 = t + 1 \wedge v_3^2 \leq 1 \wedge v_1 = X(t) \wedge v_3 = X(v_2).$$

The common variables are  $\{v_1, v_2, v_3, t\}$ , and the equivalence relation induced by the set of equivalence classes  $\{\{v_1, v_3\}, \{v_2\}, \{t\}\}$  illustrates Lemma 2, since

$$v_1 \geq 1 \wedge v_2 = t + 1 \wedge v_3^2 \leq 1 \wedge v_1 = v_3 \wedge v_1 \neq v_2 \wedge v_3 \neq v_2$$

<sup>1</sup> A theory  $T$  with signature  $\Sigma$  is called stably infinite iff for every quantifier-free  $\Sigma$ -formula  $\phi$ , if  $F$  is  $T$ -satisfiable, then there exists some  $T$ -interpretation that satisfies  $F$  and has a domain of infinite cardinality [27, 6].

is satisfiable in the theory of real-closed fields, and

$$v_1 = X(t) \wedge v_3 = X(v_2) \wedge v_1 = v_3 \wedge v_1 \neq v_2 \wedge v_3 \neq v_2$$

is satisfiable in the theory of uninterpreted function symbols.

The second lemma states a Hermite-like interpolation property whose proof follows from standard techniques in mathematical analysis.

► **Lemma 3.** *Let  $p$  be a function from a finite subset  $P$  of  $\mathbb{R}^n \times \mathbb{N}_0^n$  to  $\mathbb{R}$ . Then there exists a smooth function  $F : \mathbb{R}^n \rightarrow \mathbb{R}$  s.t. for every  $(x, d) \in P$ ,  $(D^d F)(x) = p(x, d)$ .*

**Proof.** Let  $X$  be the set  $\{x \mid (x, d) \in P\}$ . This set is finite, and hence the elements of  $X$  are isolated. For each  $c \in X$ , construct a smooth function  $f_c$  which for all  $d$  with  $(c, d) \in P$ ,  $(D^d f_c)(c) = p(c, d)$ . Let  $F : \mathbb{R}^n \rightarrow \mathbb{R}$  be such that for all  $x \in \mathbb{R}^n$ ,  $F(x) = \sum_{c \in X} B_c(f_c(x))$ , where  $B_c$  is a smooth function that is equal to the identity function in a sufficiently small neighborhood of  $c$ , and zero around all other elements of  $X$  (i.e., a so-called bump function). Then  $F$  satisfies the desired property. ◀

Now we return to the proof of Theorem 1:

**Proof.** To prove the  $\Rightarrow$  direction, we assume a variable assignment  $\alpha$  that  $\mathcal{F}$ -satisfies  $\phi$  and construct an interpretation that satisfies both the axioms of  $\mathcal{RU}$  and the formula  $\tau(\phi)$ . The interpretation is based on the structure of the real numbers, interprets the symbols  $\{0, 1, +, \times, \leq, <, \geq, >\}$  in the usual mathematical way, interprets the function symbols introduced by the translation  $\tau$  as the corresponding smooth real-valued functions given by  $\alpha$  and their respective derivatives, and assigns to the variables of  $\tau(\phi)$  the corresponding real values given by  $\alpha$ . The result satisfies the formula  $\phi$  by construction and satisfies the axioms of  $\mathcal{RU}$  since the real numbers are an instance of the theory of real closed fields,

We are left with proving the  $\Leftarrow$  direction. For this, we assume that  $\tau(\phi)$  is  $\mathcal{RU}$ -satisfiable, and build an assignment that satisfies  $\phi$ , assigning to each variable of  $\phi$  an element of the domain of its respective sort (i.e., either a real number and or a smooth real function).

Observe that both the theory of real closed fields and the theory of uninterpreted function symbols are stably infinite. Hence we can apply the Nelson-Oppen method. Let the formulas  $\pi_R$  and  $\pi_U$  be the result of applying the variable abstraction phase of the Nelson-Oppen method to  $\tau(\phi)$ . Hence,  $\pi_R$  is a formula in the language of real closed fields, and  $\pi_U$  a formula in the language of uninterpreted function symbols s.t.  $\pi_R \wedge \pi_U$  is  $\mathcal{RU}$ -equi-satisfiable with  $\tau(\phi)$ . Let  $V$  be the common variables of  $\pi_R$  and  $\pi_U$ , and let  $E$  be the equivalence relation on  $V$  ensured by Lemma 2. Then  $\pi_R \wedge \rho(V, E)$  is satisfiable in the theory of real closed fields and  $\rho(V, E) \wedge \pi_U$  in the theory of uninterpreted function symbols. Note that those theories are defined axiomatically, and hence satisfying interpretations do not necessarily have to be based on real numbers.

The theory of real closed fields is complete, and hence all its models are elementary equivalent. As a consequence, there is an interpretation  $I_R$  that satisfies  $\pi_R \wedge \rho(V, E)$  and assigns real numbers to all variables.

Also  $\rho(V, E) \wedge \pi_U$  has a satisfying interpretation. However, since the theory of uninterpreted symbols is not complete, we need a more involved construction to come up with an interpretation assigning real numbers and real-valued functions.

We observe that for a formula that is satisfiable in the theory of uninterpreted function symbols, the congruence closure algorithm [28] constructs a satisfying interpretation. Its domain is formed by equivalence classes  $T_\sim$  of the set of sub-terms  $T$  of the given formula. This domain is finite since the set  $T$  is finite. Moreover, each equivalence class contains only

finitely many terms. Let  $I_U$  be such an interpretation satisfying the formula  $\rho(V, E) \wedge \pi_U$ . Observe that the arrangement  $\rho(V, E)$  ensures that all variables shared by  $\pi_R$  and  $\pi_U$  and belonging to the same equivalence class, must have the same value in  $I_R$ .

We will combine the interpretations  $I_R$  and  $I_U$  into an interpretation  $I$  that satisfies  $\tau(\phi)$  and, in addition, uses the real numbers as its domain. Hence, we will translate the elements in the domain of  $I_U$  to real numbers, and extend them to real-valued functions corresponding to the uninterpreted function symbols.

We will now define a function  $r$  assigning to each equivalence class in  $T_{\sim}$  a distinct real number. Let this function  $r$  be such that it assigns to each equivalence class containing a variable shared by  $\pi_R$  and  $\pi_U$  the value of this variable in  $I_R$  (as we have observed, this is unique over all such variables belonging to the same equivalence class), and to all other equivalence classes a further, distinct real number. Let  $r' : T \rightarrow \mathbb{R}$  s.t. for every term  $t \in T$ ,  $r'(t)$  is the real number that  $r$  assigns to the equivalence class containing  $t$ .

Based on this, let  $I$  be the following interpretation which assigns real numbers to all variables in  $\tau(\phi)$  and partial real functions to the function symbols in  $\tau(\phi)$ :

- for every variable  $x$  occurring in  $\pi_R$ ,  $I(x) := I_R(x)$ ,
- for every variable  $x$  occurring in  $\pi_U$ ,  $I(x) := r'(x)$ ,
- for every function symbol  $X$  of arity  $k$ , let  $I(X)$  be the partial function such that for every term of the form  $X(t_1, \dots, t_k)$  in  $T$ ,  $I(X)(r'(t_1), \dots, r'(t_k)) := r'(X(t_1, \dots, t_k))$ , and  $I(X)$  is undefined for all other values.

The two following observations make this definition well-formed:

- The first two items overlap. This is no problem since for shared variables,  $I(x)$  and  $r'(x)$  coincide.
- The definition in the third item is unique since due to the congruence axioms of the theory of free function symbols, for all  $t_1, \dots, t_k, t'_1, \dots, t'_k \in T$ ,  $r'(t_1) = r'(t'_1), \dots, r'(t_k) = r'(t'_k)$  implies  $r'(X(t_1, \dots, t_k)) = r'(X(t'_1, \dots, t'_k))$ .

We will now build a variable assignment  $\alpha$  from  $I$  such that  $\alpha \models \phi$ . For every scalar variable  $x$ ,  $\alpha(x) := I(x)$ . For every function variable  $V$ ,  $\alpha(V)$  will be a smooth real-valued function whose values coincide with the values of the partial function  $I(V)$  on all points where this partial function has a defined value, and whose derivatives coincides with the values of the corresponding partial function  $I(V_{i_1, \dots, i_n})$  on all points where the latter has a defined value. Such a function exists due to Lemma 3, and it satisfies the formula  $\phi$ . ◀

To illustrate the theorem, we continue with the example from above. The real part of the formula is satisfiable, for example by  $\{v_1 \mapsto 1, v_2 \mapsto 7, v_3 \mapsto 1, t \mapsto 6\}$ . Applying the congruence closure algorithm to the part with uninterpreted function symbols, we work with the set of sub-terms  $T = \{v_1, v_2, v_3, t, X(t), X(v_2)\}$ . The result of the congruence closure algorithm is the equivalence relation  $\{\{v_1, v_3, X(v_2), X(t)\}, \{t\}, \{v_2\}\}$ . Hence  $r'$  is  $\{v_1 \mapsto 1, v_3 \mapsto 1, X(v_2) \mapsto 1, X(t) \mapsto 1, t \mapsto 6, v_2 \mapsto 7\}$ . Since every variable in  $\pi_U$  also occurs in  $\pi_R$ , the interpretation  $I$  can simply agree with  $r'$  on the scalar variables. Moreover, it assigns to the function variable  $X$  the partial function  $\{7 \mapsto 1, 6 \mapsto 1\}$ . The corresponding assignment  $\alpha$  assigns to the scalar variables the same real values as  $I$ , and assigns to  $X$  a smooth interpolation of the partial function  $\{7 \mapsto 1, 6 \mapsto 1\}$ . For example, this could be the constant function that has the value 1, everywhere.

Since a disjunction of formulas is satisfiable, if one of the constituting disjuncts is satisfiable, we get:

► **Corollary 4.** *The quantifier free, function-algebraic theory of real functions is decidable.*

The proof of Theorem 1 also shows how to compute satisfying assignments: After checking the satisfiability of  $\tau(\phi)$  using the congruence closure algorithm and the Nelson-Oppen combination procedure, construct the variable assignment  $\alpha$  defined in the proof.

Note that the concluding building block of the proof of Theorem 1 is Lemma 3. Any analogous lemma that ensures stronger properties of the constructed functions results in a corresponding strengthening of Theorem 1. For example, we could also be interested in constructing functions that generalize the constraints given by the input formula as much as possible, maximizing certain regularity properties [14].

## 5 Scalar Quantification

We will now allow arbitrary quantification on scalar variables. We will still require formulas to be function-algebraic and do not allow quantification on function variables. An example is

$$\forall t \in [0, 10] \exists t' \in [0, 10] . \text{app}(X, t, 2t)^2 + 1 \geq \text{app}(\partial_2 X, t', t'),$$

where  $X \in \mathcal{F}_2$  and the interval bounds on variables represent the obvious abbreviations. Many problems resulting from the synthesis of correctness certificates for continuous systems (e.g., Lyapunov function [22], barrier certificates [30] and their generalizations [29, 20]), belong to this class.

In Subsection 5.1, we will introduce a method for checking satisfiability under the condition that the functional variables are instantiated to fixed, user-provided polynomials. In Subsection 5.2, we will introduce a robustness property of formulas that will allow us to characterize solvability of formulas. In Subsection 5.3, we will introduce a systematic method for applying the satisfiability check from Subsection 5.1 and ensuring that it will succeed for all formulas satisfying the robustness property. In Subsection 5.4, we will discuss the relevance of the method for practical computation.

### 5.1 Check Satisfiability Under Polynomial Instantiations

The method for checking satisfiability of a formula that we introduce in this sub-section depends on an instantiation of its functional variables to polynomials with rational coefficients. This will allow us to rewrite the formula into a formula in the language of the theory of real-closed fields which is decidable. In this sub-section, we still assume these polynomials to be given (e.g., chosen by the user), and drop this assumption later.

► **Definition 5.** *We call a function  $\pi$  that assigns to every function variable of sort  $\mathcal{F}_n$  a polynomial with rational coefficients in the variables  $t_1, \dots, t_n$  a polynomial assignment. Moreover, we call a pair consisting of a formula and a polynomial assignment an instantiated formula.*

The intuition is that the polynomial assignment  $\pi$  in an instantiated formula  $(\phi, \pi)$  instantiates each function variable in  $\phi$  to the respective polynomial assigned by  $\pi$ . Let us define the following rules on instantiated formulas  $(\phi, \pi)$ :

- **varep:** Rename multiple occurrences of the same function variable in  $\phi$  by fresh function variables, and extend  $\pi$  to the new variables in such a way that it assigns the same polynomial to each new variable as to its original one.
- **$\partial$ -elim:** Replace a sub-term of  $\phi$  of the form  $\partial_i X$ , where  $X$  is a function variable, by  $X$  and  $\pi$  by  $\pi'$  that is identical to  $\pi$  except that it assigns the result of symbolic differentiation of the polynomial  $\pi(X)$  in its  $i$ -th argument to the function variable  $X$ .

■ **Table 1** Polynomial Instantiation.

rule	formula	polynomial assignment
varsep	$\forall p \forall q . app(X, q) + p^2 app(\partial_1 X, r) app(Y, q) \geq 0$	$\{X \mapsto t^2 + 1, Y \mapsto t\}$
$\partial$ -elim	$\forall p \forall q . app(X, q) + p^2 app(\partial_1 X', r) app(Y, q) \geq 0$	$\{X \mapsto t^2 + 1, X' \mapsto t^2 + 1, Y \mapsto t\}$
$app$ -elim	$\forall p \forall q . app(X, q) + p^2 app(X', r) app(Y, q) \geq 0$	$\{X \mapsto t^2 + 1, X' \mapsto 2t, Y \mapsto t\}$
	$\forall p \forall q . q^2 + 1 + 2p^2 r q \geq 0$	$\{X \mapsto t^2 + 1, X' \mapsto 2t, Y \mapsto t\}$

- **$app$ -elim:** Replace a sub-term of  $\phi$  of the form  $app(X, \hat{t}_1, \dots, \hat{t}_n)$ , where  $X$  is a function variable, and the terms  $\hat{t}_1, \dots, \hat{t}_n$  do not contain any  $app$ -operator, by the result of evaluating the polynomial  $\pi(X)$  using the values  $\hat{t}_1, \dots, \hat{t}_n$  for the respective variables  $t_1, \dots, t_n$ .

Now apply first the rule  $varsep$ , and then iterate applying the elimination rules until they cannot be applied any more. This process must terminate since every application of an elimination rule decreases the total number of  $\partial$ - and  $app$ -operators by one. Moreover, the result must be unique since the only possible alternative choices of the rules relate to independent sub-formulas. So denote by  $\Pi_\pi(\phi)$  the formula  $\phi'$  where  $(\phi', \pi')$  is the final result of the described rule-application process.

Table 1 shows the results of the individual steps of the process of forming  $\Pi_\pi(\phi)$  for an example of an instantiated formula with  $X, Y \in \mathcal{F}_1, p, q, r \in \mathcal{R}$ , where the result  $\Pi_\pi(\phi)$  can be seen at the bottom of the column “formula”.

Polynomial evaluation completely eliminates any function variables or operators:

- **Lemma 6.** *For every instantiated formula  $(\phi, \pi)$ ,  $\Pi_\pi(\phi)$  does not contain any function variable, and hence it also does not contain any  $app$ - or  $diff$ -operator.*

**Proof.** Function variables can only occur as arguments to  $diff$  and  $app$  operators. In such a situation, the rules  $\partial$ -elim and  $app$ -elim are applicable, and hence, such a formula cannot be the result of  $\Pi_\pi(\phi)$ . ◀

Therefore, if  $\phi$  is function-algebraic,  $\Pi_\pi(\phi)$  is a formula in the language of real-closed fields, which is decidable [38]. Moreover, instantiated formulas can be used for proving satisfiability:

- **Theorem 7.** *For every instantiated formula  $(\phi, \pi)$ ,  $\Pi_\pi(\phi)$  is satisfiable iff  $\pi \models \exists_{\mathcal{R}} \phi$ , where  $\exists_{\mathcal{R}}$  denotes the existential closure of the formula wrt. the scalar variables.*

**Proof.** For an instantiated formula  $(\phi, \pi)$ , let  $\rho(\phi, \pi)$  be the formula

$$\exists_{\mathcal{R}} \phi \wedge \bigwedge_{X \in \mathcal{V}_F} \forall t_1, \dots, t_n . app(X, t_1, \dots, t_n) = \pi(X).$$

Observe that  $\pi \models \exists_{\mathcal{R}} \phi$  iff  $\rho(\phi, \pi)$  is satisfiable. Moreover, every element of the sequence  $\rho(\phi_1, \pi_1), \dots, \rho(\phi_n, \pi_n)$  with  $(\phi_1, \pi_1) = (\phi, \pi)$ ,  $\phi_n = \Pi_\pi(\phi)$ , and each  $(\phi_i, \pi_i)$ ,  $i \in \{2, \dots, n\}$  being the result of the application of a rewrite rule to  $(\phi_{i-1}, \pi_{i-1})$ , is equi-satisfiable. Finally  $\rho(\phi_n, \pi_n)$  and  $\phi_n = \Pi_\pi(\phi)$  are equi-satisfiable since  $\Pi_\pi(\phi)$  does not contain any function variable. ◀

So we have reduced the satisfiability checking problem to the problem of finding a polynomial assignment  $\pi$  for which  $\Pi_\pi(\phi)$  is satisfiable. However, for some satisfiable formulas, the search for such a polynomial assignment is bound to fail. This can be easily seen on the simple initial value problem

$$\text{app}(X, 0) = 1 \wedge \forall t . t \geq 0 \Rightarrow \text{app}(\dot{X}, t) = \text{app}(X, t)$$

that is satisfiable, but not by any polynomial assignment (the only solution of the given initial value problem is the exponential function).

## 5.2 Robust Satisfiability

Even though differential equations such as  $\dot{x} = x$  (in our notation:  $\forall t . \text{app}(\dot{X}, t) = \text{app}(X, t)$ ) are ubiquitous in mathematics, they are highly idealized objects: In practice, no real physical system will satisfy such an equation precisely, and concrete differential equations can only be used in applications after introducing many simplifying assumptions that are part of the daily bread of practical engineering. However, this also makes it necessary for engineers to assess the consequences of such simplifications. Despite the existence of powerful deductive verification techniques [29, 15], in practice, differential equations are still solved by algorithms [19] that produce approximation errors both due to discretization and due to floating point computation. The reliability of the whole process depends essentially on the fact that the error made by the solver does not dominate the error made by simplifying assumptions. This is a major complication, that could be avoided if solvers could conservatively bound the produced errors. For the concrete example  $\dot{x} = x$ , it would be very useful, if a solver could – instead of solving the differential equation approximately – guarantee the solution of  $x - \epsilon \leq \dot{x} \leq x + \epsilon$  within a compact set, for a small constant  $\epsilon > 0$ . In this section, we will formally characterize such situations and show that in such cases, a formally correct satisfiability check is not only possible, but that we can even guarantee its success.

For being able to measure the distance between variable assignments, we will adjoin metrics to the set of variable assignments  $\aleph$ , making the pair  $(\aleph, d)$  a metric space. These metrics will be parametric in a family of compact sets  $K_n \subseteq \mathbb{R}^n, n \in \mathbb{N}$  which we will call *domain of interest*. We will denote this dependence on the domain of interest by an index, writing  $d_K$  for the metric associated to domain of interest  $K$ . We will call such a metric on  $\aleph$  a *variable assignment metric*.

► **Definition 8.** *A formula  $\phi$  is semantically robustly satisfiable wrt. a variable assignment metric  $d$  iff there is a variable assignment  $\alpha \in \aleph$  and an  $\epsilon > 0$  (that we call the robustness margin) such that for every  $\alpha'$  with  $d_K(\alpha, \alpha') < \epsilon$ ,  $\alpha' \models \phi$ .*

Note that unlike similar definitions [17, 33], this definition only depends on the semantics of a given formula, but not on its syntax, and hence is invariant wrt. equivalence transformations. We will later see that this is made possible by the fact that we restrict ourselves to operations on real numbers allowed by the decidable theory of real closed fields.

We will usually use metrics induced by some norm, and so we will call a formula robustly satisfiable wrt. a norm  $\|\cdot\|_K$  iff it is robustly satisfiable wrt. the metric  $d_K(x, x') = \|x - x'\|_K$ . Given metrics  $d^{\mathcal{T}}$  on  $\mathcal{T}$ , where  $\mathcal{T} \in \{\mathcal{R}, \mathcal{F}_1, \dots\}$ , we define their extension to variable assignments element-wise. So, for  $\alpha, \alpha' \in \aleph$ ,

$$d_K(\alpha, \alpha') := \max_{\mathcal{T} \in \{\mathcal{R}, \mathcal{F}_1, \dots\}} \max_{v \in \mathcal{V}_{\mathcal{T}}} d_K^{\mathcal{T}}(\alpha(v), \alpha'(v)).$$

Here, we will usually use a family of metrics on function variables of all dimensions. If  $d^{\mathcal{R}}$  is a metric on  $\mathbb{R}$  and  $d^{\mathcal{F}}$  such a family of metrics on smooth functions  $\mathbb{R}^i \rightarrow \mathbb{R}, i \in \mathbb{N}$ , then we will denote this extension to variable assignments by  $d^{\mathcal{R}} \times d^{\mathcal{F}}$ .

On real-numbers we will use the discrete metric  $d^{\mathcal{R}}(x, y) := \begin{cases} 1, & \text{if } x \neq y, \\ 0, & \text{if } x = y. \end{cases}$

The metric on functions will be based on a norm measuring the size of a given function and of its derivatives. For a function  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ , at least  $k$ -times differentiable, let

$$\|F\|_K^k := \max_{|\beta| \leq k} \inf_{x \in K_n} |(D^\beta F)(x)|.$$

We denote the metric induced by this norm  $\|\cdot\|_K^k$  by  $d_K^k$ . Here are some examples:

- $\forall t \in [0, 1] . \text{app}(X, t) - 0.1 \leq \text{app}(\dot{X}, t) \wedge \text{app}(\dot{X}, t) \leq \text{app}(X, t) + 0.1$ , with  $x \in \mathcal{F}_1$  is *not* robustly satisfiable wrt. the norm  $\|\cdot\|_{[0,1]}^0$ , since that norm does not constrain any derivative of  $x$ . However, it is robustly satisfiable wrt.  $\|\cdot\|_{[0,1]}^1$  since, for example, every function with maximal distance 0.01 from the exponential function wrt.  $\|\cdot\|_{[0,1]}^1$  satisfies the formula.
- $\forall t \in [0, 1] . \text{app}(\dot{X}, t) = \text{app}(X, 0)$  is not robustly satisfiable wrt.  $\|\cdot\|_{[0,1]}^0$ , since for every function satisfying the formula adding the term  $\epsilon t$  to the function results in a function not satisfying it.
- The formula  $\forall t . \text{app}(X, t) \geq 0$ , while satisfiable, is not robustly satisfiable wrt. the norm  $\|\cdot\|_{[0,1]}^0$ , since this norm only restricts the value of functions in the domain of interest  $[0, 1]$ . Due to this, for every variable assignment  $\alpha$  satisfying the formula, there is an  $\alpha'$  with  $d(\alpha, \alpha') = 0$  that does not satisfy the formula: Simply choose an  $\alpha'$  that is identical to  $\alpha$  on  $[0, 1]$  but reaches a negative value outside of this interval. In contrast to that, the formula  $\forall t \in [0, 1] . \text{app}(X, t) \geq 0$ , is robustly satisfiable which explains the importance of bounds on quantified variables for ensuring robustness.

### 5.3 Robust Completeness

We will now introduce a systematic method that checks satisfiability using the test from Subsection 5.1 and that will succeed for all formulas satisfying the robustness property from Subsection 5.2. We will use the fact that one can approximate smooth functions on compact domains arbitrarily closely by polynomials. For this, recall that a subset  $X'$  of a metric space  $(X, d)$  is dense in  $(X, d)$  iff for every  $x \in X$  and  $\varepsilon > 0$  there is an  $x' \in X'$  with  $d(x, x') < \varepsilon$ .

► **Lemma 9.** *Let  $\phi$  be a formula,  $d$  a variable assignment metric and  $\aleph' \subseteq \aleph$  s.t.  $\aleph'$  is dense in  $(\aleph, d)$ . Then every formula that is semantically robustly satisfiable wrt. the metric  $d$  has a satisfying assignment from  $\aleph'$ .*

**Proof.** Assume that  $\phi$  is semantically robustly satisfiable wrt.  $d$ . Then there is a variable assignment  $\alpha \in \aleph$  and an  $\varepsilon > 0$  such that for every  $\alpha'$  with  $d(\alpha, \alpha') < \varepsilon$ ,  $\alpha' \models \phi$ . Since  $\aleph'$  is dense in  $(\aleph, d)$ , there is an  $\alpha' \in \aleph'$  with  $d(\alpha, \alpha') < \varepsilon$ . Hence  $\alpha'$  is within the robustness margin of  $\alpha$ , and hence  $\alpha' \models \phi$ . ◀

Now we observe:

► **Lemma 10.** *For every  $k \in \mathbb{N}_0$ , compact  $K \subseteq \mathbb{R}^n$ , the set of  $n$ -dimension polynomial functions with rational coefficients is dense in the set of  $n$ -dimensional polynomial functions with real coefficients wrt. the metric  $d_K^k$ .*

**Proof.** Let  $P$  be a polynomial function with real coefficients and  $\varepsilon > 0$ . We will prove that there is a polynomial  $P'$  with rational coefficients such that  $d_K^k(P, P') < \varepsilon$ , that is  $\max_{|\beta| \leq k} \inf_{x \in K} |(D^\beta P)(x) - (D^\beta P')(x)| < \varepsilon$ . During this, we will separate any given polynomial  $P$  into its vectors of coefficients  $C_P$  and monomials  $M_P$ . Hence  $P = C_P^T M_P$ .

Let  $m = \max_{|\beta| \leq k} \inf_{x \in K} \|M_{D^\beta P}(x)\|$  with  $\|\cdot\|$  denoting the Euclidean metric. The value  $m$  is finite since  $K$  is compact. Let  $P'$  be a polynomial with rational coefficients s.t.  $M_P(x) = M_{P'}(x)$  and s.t.  $\max_{|\beta| \leq k} \|C_{D^\beta P} - C_{D^\beta P'}\| < \frac{\varepsilon}{m}$ . Now, due to Cauchy-Schwarz,

$$\begin{aligned} \max_{|\beta| \leq k} \inf_{x \in K} |(D^\beta P)(x) - (D^\beta P')(x)| &= \max_{|\beta| \leq k} \inf_{x \in K} |(C_{D^\beta P} - C_{D^\beta P'})^T M_{D^\beta P}(x)| \leq \\ & \max_{|\beta| \leq k} \inf_{x \in K} \|C_{D^\beta P} - C_{D^\beta P'}\| \|M_{D^\beta P}(x)\| < \frac{\varepsilon}{m} m = \varepsilon \quad \blacktriangleleft \end{aligned}$$

Moreover, the classical Stone-Weierstrass theorem generalizes to  $d_K^k$ , that is, the polynomials with real coefficients  $P(\mathbb{R})$  are dense in the set of  $C_k$ -real functions on any compact set  $K$  wrt.  $d_K^k$  [40, 26]. This allows us to conclude:

► **Lemma 11.** *For every  $k \in \mathbb{N}_0$  and compact  $K \subseteq \mathbb{R}^n$ , the set of  $n$ -dimensional polynomial functions with rational coefficients is dense in the metric space  $(\mathbb{N}, d_K^k)$ .*

Putting everything together, we get :

► **Theorem 12.** *For every  $k \in \mathbb{N}_0$  and and family of compact sets  $K_n \subseteq \mathbb{R}^n$ ,  $n \in \mathbb{N}$ , there is an algorithm for checking satisfiability of function-algebraic formulas that terminates for every input formula that is semantically robustly satisfiable wrt. the metric  $d^\# \times d_K^k$ .*

**Proof.** Assume a semantically robustly satisfiable function-algebraic formula  $\phi$ . Since by Lemma 11, the set of variable assignments assigning real values to scalar variables and polynomials with rational coefficients to function variables is dense in the metric space  $(\mathbb{N}, d^\# \times d_K^k)$ , by Lemma 9, there is an assignment  $\alpha \in \mathbb{N}$  that assigns polynomials with rational coefficients to function variables and that satisfies  $\phi$ . The restriction of  $\alpha$  to a polynomial assignment  $\alpha_\pi$  satisfies  $\exists_{\mathcal{R}} \phi$ .

Due to Theorem 7 we can algorithmically check whether for a given polynomial assignment  $\pi$ ,  $\pi \models \exists_{\mathcal{R}} \phi$ . Moreover, the set of polynomial assignments is recursively enumerable. Hence, an algorithm that enumerates its elements, checking for each element  $\pi$  whether  $\Pi_\pi(\phi)$  is satisfiable in the theory of real closed fields, will eventually find  $\alpha_\pi$ , and hence terminate proving that  $\phi$  is satisfiable. ◀

Note however, that we do not know how to check a given formula for robustness. Hence, for a given formula we do not know a-priori whether the enumeration algorithm from the proof of Theorem 12 will terminate. We just know that it will terminate *under the assumption* that the formula is robust.

## 5.4 Practical Computation – Templates

Of course, the algorithm from the proof of Theorem 12 is hopelessly inefficient in practice. Still, our approach may provide useful practical insight. In practice, problems of the kind studied here are solved in many, often distant areas [36, 31, 34, 35]. A common approach is to restrict the set of potential solutions to a fixed class of functions given by a parameterized expression (sometimes also called a *template*), and then searching for values for the parameters such that the result of instantiating the parameters by those values represents a solution to the problem.

There are two main classes of templates that are often used here. The first class are templates given by complex expressions, often called neural networks. The second class are polynomials whose coefficients are parameters which allows many methods to exploit the fact that polynomials are linear in their coefficients. If the given template polynomial

does not represent a solution, one can increase the degree of the polynomial. The resulting loop amounts to an enumeration of all polynomials with real coefficients. Our approach (1) formally justifies such algorithms showing that such a loop must terminate for all robust inputs, and (2) generalizes such algorithms to all formulas belonging to the language used in this paper.

## 6 Conclusion

We have developed a framework for decision procedures for a predicate logical theory formalizing a notion that is central to mathematics, computer science, and many other scientific fields – real-valued functions. Our long-term vision is to replace the need for research on application-specific automated reasoning techniques for smooth real-valued functions by a common framework that results in tools that can be used out-of-the-box in a similar way as decision procedure for common first-order theories in the frame of SMT solvers [3].

---

### References

- 1 Charu C Aggarwal. Neural networks and deep learning. *Springer*, 10:978–3, 2018.
- 2 Eugene Asarin and Ahmed Bouajjani. Perturbed Turing machines and hybrid systems. In *Proc. LICS'01*, pages 269–278, 2001.
- 3 Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*. Springer International Publishing, 2018.
- 4 Olivier Bournez and Manuel L. Campagnolo. A survey on continuous time computations. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms*, pages 383–423. Springer New York, 2008.
- 5 Olivier Bournez and Amaury Pouly. *Handbook of Computability and Complexity in Analysis*, chapter A Survey on Analog Models of Computation, pages 173–226. Springer, 2018.
- 6 Aaron Bradley and Zohar Manna. *The Calculus of Computation*. Springer, 2007.
- 7 Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Transactions on Computational Logic (TOCL)*, 19(3):1–52, 2018.
- 8 Pieter Collins and Daniel Silva Graça. Effective computability of solutions of differential inclusions the ten thousand monkeys approach. *J. Univers. Comput. Sci.*, 15(6):1162–1185, 2009.
- 9 Pieter Collins, Milad Niqui, and Nathalie Revol. A validated real function calculus. *Mathematics in Computer Science*, 5(4):437–467, 2011.
- 10 Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of hybrid systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*. Springer International Publishing, 2018.
- 11 T. A. Driscoll, N. Hale, and L. N. Trefethen. *Chebfun guide*. Pafnuty Publications, Oxford, 2014.
- 12 Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *Automated Technology for Verification and Analysis*, volume 5311 of *LNCS*, 2008.
- 13 Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design – A Practical Guide*. Academic Press, 1988.
- 14 Charles Fefferman and Arie Israel. *Fitting Smooth Functions to Data*. AMS, 2020.

- 15 Simon Foster, Jonathan Julián Huerta y Munive, Mario Gleirscher, and Georg Struth. Hybrid systems verification with Isabelle/HOL: Simpler syntax, better models, faster proofs. In *International Symposium on Formal Methods*, pages 367–386. Springer, 2021.
- 16 Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, number 1683 in LNCS. Springer, 1999.
- 17 Sicun Gao, Jeremy Avigad, and Edmund Clarke.  $\delta$ -decidability over the reals. In *LICS*, pages 305–314. IEEE, 2012.
- 18 Sicun Gao, Soonho Kong, and Edmund M Clarke. Satisfiability modulo ODEs. In *2013 Formal Methods in Computer-Aided Design*, pages 105–112. IEEE, 2013.
- 19 Ernst Hairer, Syvert Paul Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I*. Springer-Verlag, 1987.
- 20 Hyejin Han, Mohamed Maghenem, and Ricardo G. Sanfelice. Certifying the LTL formula p until q in hybrid systems. *IEEE Transactions on Automatic Control*, 68(7), 2023.
- 21 Timothy J. Hickey. Analytic constraint solving and interval arithmetic. In *Proc. of the 27th ACM SIGACT-SIGPLAN Symp. on Principles of Progr. Lang.*, pages 338–351. ACM Press, 2000.
- 22 Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 3rd edition, 2002.
- 23 Ker-I Ko. Computational complexity of real functions. In *Complexity Theory of Real Functions*, Progress in Theoretical Computer Science, pages 40–70. Birkhäuser Boston, 1991.
- 24 Peter D. Lax. *Functional Analysis*. Wiley-Interscience, 2002.
- 25 Angus Macintyre and A.J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana – About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.
- 26 Raghavan Narasimhan. *Analysis on Real and Complex Manifolds*. Elsevier, 2nd edition edition, 1973.
- 27 Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
- 28 Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, April 1980.
- 29 André Platzer. *Logical Foundations of Cyber-Physical Systems*, volume 662. Springer, 2018.
- 30 Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Rajeev Alur and George J. Pappas, editors, *HSCC'04*, number 2993 in LNCS. Springer, 2004.
- 31 Stephen Prajna and Anders Rantzer. Convex programs for temporal verification of nonlinear dynamical systems. *SIAM Journal on Control and Optimization*, 46(3):999–1021, 2007.
- 32 Stefan Ratschan. Continuous first-order constraint satisfaction. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, number 2385 in LNCS, pages 181–195. Springer, 2002.
- 33 Stefan Ratschan. Quantified constraints under perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.
- 34 Stefan Ratschan. Simulation based computation of certificates for safety of dynamical systems. In Alessandro Abate and Gilles Geeraerts, editors, *Formal Modeling and Analysis of Timed Systems: 15th International Conference, FORMATS 2017*, volume 10419, pages 303–317. Springer International Publishing, 2017.
- 35 Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning control Lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, pages 1–33, 2018.
- 36 Junuthula Narasimha Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Education, 2019.
- 37 Robert M Solovay, RD Arthan, and John Harrison. Some new results on decidability for elementary algebra and geometry. *Annals of Pure and Applied Logic*, 163(12):1765–1802, 2012.

- 38 A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press, Berkeley, 1951.
- 39 Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In *Frontiers of Combining Systems*, pages 103–119. Springer, 1996.
- 40 A. Yu. Veretennikov and E. V. Veretennikova. On partial derivatives of multivariate Bernstein polynomials. *Siberian Advances in Mathematics*, 26(4):294–305, 2016.
- 41 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.



# A Polynomial-Time Algorithm for MCS Partial Search Order on Chordal Graphs

Guozhen Rong ✉

Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation,  
Changsha University of Science and Technology, China

Yongjie Yang ✉ 

Chair of Economic Theory, Saarland University, Saarbrücken, Germany

Wenjun Li ✉

Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation,  
Changsha University of Science and Technology, China

---

## Abstract

We study the partial search order problem (PSOP) proposed recently by Scheffler [WG 2022]. Given a graph  $G$  together with a partial order on the set of vertices of  $G$ , this problem determines if there is an  $\mathcal{S}$ -ordering that is consistent with the given partial order, where  $\mathcal{S}$  is a graph search paradigm like BFS, DFS, etc. This problem naturally generalizes the end-vertex problem which has received much attention over the past few years. It also generalizes the so-called  $\mathcal{F}$ -tree recognition problem which has just been studied in the literature recently. Our main contribution is a polynomial-time dynamic programming algorithm for the PSOP of the maximum cardinality search (MCS) restricted to chordal graphs. This resolves one of the most intriguing open questions left in the work of Scheffler [WG 2022]. To obtain our result, we propose the notion of layer structure and study numerous related structural properties which might be of independent interest.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial algorithms; Theory of computation → Graph algorithms analysis

**Keywords and phrases** partial search order, maximum cardinality search, chordal graphs, clique graphs, dynamic programming

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.77

**Related Version** *Full Version*: <https://arxiv.org/pdf/2212.04880.pdf>

**Funding** This paper was supported by the Research Foundation of Education Bureau of Hunan Province under grant 21B0305 and Natural Science Foundation of Hunan Province of China under grant 2022JJ30620.

**Acknowledgements** The authors thank the anonymous reviewers of MFCS 2023 for their careful reading and instructive comments.

## 1 Introduction

Graph search paradigms are pervasive in algorithms for innumerable graph problems. In addition to the most popular paradigms breadth-first search (BFS) and depth-first search (DFS), several other prevalent graph search paradigms – including, for instance, lexicographic breadth-first search (LBFS), lexicographic depth-first search (LDFS), maximum cardinality search (MCS), maximal neighborhood search (MNS) – have also been extensively studied in the literature [10, 25, 28, 29]. These graph search paradigms have proved to be exclusively useful in dealing with a variety of graph problems [6, 8, 9, 22]. For instance, MCS has been successfully used in the recognition of special graphs [29], the computation of minimal separators [22], the computation of minimal triangulation of graphs [3], determining lower bounds of treewidth [5, 21], etc. In several of these algorithmic applications, last vertices



© Guozhen Rong, Yongjie Yang, and Wenjun Li;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 77; pp. 77:1–77:15

Leibniz International Proceedings in Informatics



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

visited in graphs are crucial for the correctness of the algorithms. Last visited vertices also exhibit some nice structural properties. For instance, for a cocomparability graph that is Hamiltonian, if a vertex is last visited by LDFS, then there is a Hamiltonian path starting from this vertex [1, 8]. For more concrete examples on this issue, we refer to [1, 11]. These specialities of last visited vertices inspired Corneil, Köhler, and Lanlignel [11] to put forward the end-vertex problem, in which we are given a graph and a particular vertex  $v$ , and are asked whether  $v$  can be the last visited one according to a certain graph search paradigm. Thenceforth, investigation on the end-vertex problem has flourished, resulting in the complexity of the problem for both general graphs and many special graphs such as chordal graphs, split graphs, interval graphs, bipartite graphs, etc., being substantially established [1, 24, 32]. For a comprehensive summary of the recent progress, we refer to [16].

A closely related problem is the search tree recognition problem which has a relatively longer history [17, 18, 20]. This problem determines if a given spanning tree of a graph can be obtained via a traversal of the graph by a certain search paradigm. This problem comes natural for some search paradigms like BFS and DFS, since they not only output an ordering but also generate a spanning tree during the search. However, it is ill-defined for some other search paradigms like MCS and MNS. Aiming at overcoming the plight, Beisegel et al. [2] introduced the notions of  $\mathcal{F}$ -tree and  $\mathcal{L}$ -tree ( $\mathcal{F}$  and  $\mathcal{L}$  respectively stand for “first” and “last”). Particularly, given an ordering  $\sigma$  of the vertices of a graph, the  $\mathcal{F}$ -tree (respectively,  $\mathcal{L}$ -tree) is a spanning tree of the graph so that every vertex  $v$  other than the first one in  $\sigma$  is adjacent to its first (respectively, last) neighbor appearing before  $v$  in  $\sigma$ . BFS-trees and DFS-trees are  $\mathcal{F}$ -trees and  $\mathcal{L}$ -trees of BFS and DFS, respectively. Having these notions, Beisegel et al. [2] studied the complexity of the  $\mathcal{F}$ -tree recognition problem and the  $\mathcal{L}$ -tree recognition problem of the above-mentioned search paradigms for both general graphs and many special graph classes. Very recently, Scheffler [27] complemented these results by showing that the  $\mathcal{L}$ -tree recognition problem of BFS restricted to bipartite graphs, and the  $\mathcal{F}$ -tree recognition problem of DFS restricted to chordal graphs and chordal bipartite graphs are NP-hard, standing in a strong contrast to the polynomial-time solvability of the  $\mathcal{F}$ -tree recognition problem of BFS and the  $\mathcal{L}$ -tree recognition problem of DFS in general [18, 20].

From the known results, Scheffler [26] discerned that the complexity of the end-vertex problem and the complexity of the  $\mathcal{F}$ -tree recognition problem seemed to be somewhat connected. For instance, for LBFS, MCS, and MNS, both problems are NP-hard on weakly chordal graphs and are linear-time solvable on split graphs. Additionally, for MNS and MCS, both problems are polynomial-time solvable when restricted to chordal graphs [2, 24]. Towards a comprehensive understanding of the connection, Scheffler [26] introduced the partial search order problem (PSOP) which generalizes both the end-vertex problem and the  $\mathcal{F}$ -tree recognition problem. Given a graph  $G$  and a partial order  $R$  on the set of vertices of  $G$ , the PSOP of a search paradigm  $\mathcal{S}$  determines if  $G$  admits an  $\mathcal{S}$ -ordering which linearly extends  $R$ . Scheffler [26] derived a polynomial-time algorithm for the PSOP of LBFS restricted to chordal bipartite graphs, and a polynomial-time algorithm for the PSOP of MCS restricted to split graphs. However, whether the PSOP of MCS restricted to chordal graphs, arguably the most intriguing case, is polynomial-time solvable is unknown prior to our current work. We resolve this open question in the affirmative. To this end, we propose the notion of layer structure and study a number of structural properties which might be of independent interest. At a high level, based on the properties studied, we iteratively decompose the clique graph of a given chordal graph into what we call layer structures, handle the components (which we call units) of each layer structure separately, and utilize dynamic programming techniques to merge local solutions into a whole one.

## 2 Preliminaries

For an integer  $i$ ,  $[i]$  denotes the set of positive integers no greater than  $i$ .

### 2.1 Graphs

We follow standard notions in graph theory. For notions used but not defined in the paper, we refer to [31]. The graphs we consider are finite, undirected, and simple. Let  $G$  be a graph. The *vertex set* and *edge set* of  $G$  are denoted by  $V(G)$  and  $E(G)$ , respectively. For a vertex  $v \in V(G)$ , its *neighborhood* in  $G$ , denoted  $N_G(v)$ , is the set of vertices adjacent to  $v$  in  $G$ . A *clique* of  $G$  is a subset of pairwise adjacent vertices in  $G$ . We call a clique of  $G$  containing a vertex  $v \in V(G)$  a *v-clique*. Analogously, a clique of  $G$  containing a subset  $X \subseteq V(G)$  is called an *X-clique*. The subgraph of  $G$  induced by  $X \subseteq V(G)$  is denoted by  $G[X]$ .

A *path*  $P$  of length  $t$  is a graph with a sequence of  $t + 1$  distinct vertices  $v_1, v_2, \dots, v_{t+1}$  and with the edge set  $\{v_i v_{i+1} \mid i \in [t]\}$ . We say that  $P$  is a path between  $v_1$  and  $v_t$ , or simply call it a  $v_1$ - $v_t$  *path*. Two vertices in  $G$  are *connected* if there is a path between them in  $G$ . For  $u, v \in V(G)$ , a *u-v separator* is a subset  $X \subseteq V(G)$  so that  $u$  and  $v$  are disconnected after deleting all vertices in  $X$  from  $G$ . A *u-v separator*  $X$  is *minimal* if there are no other *u-v separators*  $X'$  such that  $X' \subsetneq X$ . The *length* of a cycle is the number of edges it contains. A *hole* is an induced cycle of length greater than three. A graph is *chordal* if it does not contain any holes as induced subgraphs.

### 2.2 The Partial Search Order Problem

A *partial order* on a set  $X$  is a reflexive, antisymmetric, and transitive binary relation on  $X$ . For ease of exposition, for a partial order  $R$ , we sometimes use  $x <_R y$  to denote  $(x, y) \in R$ . A *linear order* is a partial order that is complete. We usually write a linear order  $R$  in the format of  $(x_1, x_2, \dots, x_m)$  which means that  $(x_i, x_j) \in R$  for all  $i, j \in [m]$  such that  $i < j$ . A linear order  $R$  *extends* a partial order  $R'$  if for every  $(x, y) \in R'$  it holds that  $(x, y) \in R$ . We also call  $R$  a (*linear*) *extension* of  $R'$ . For a binary relation  $R$  on a set  $X$ , and for  $X' \subseteq X$ , we use  $R|_{X'}$  to denote  $R$  restricted to  $X'$ . For a graph search paradigm  $\mathcal{S}$  and a graph  $G$ , an  $\mathcal{S}$ -*ordering* of  $G$  is an ordering of  $V(G)$  that can be generated from an  $\mathcal{S}$  search on  $G$ . The partial search order problem of  $\mathcal{S}$ , denoted PSOP- $\mathcal{S}$ , is defined as follows.

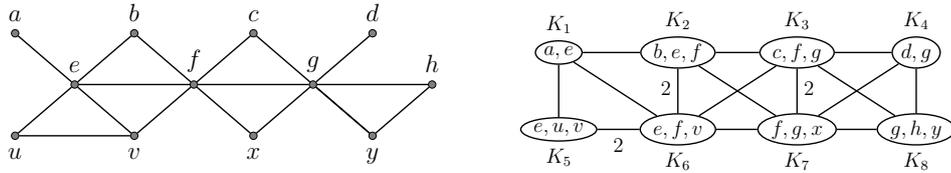
PSOP- $\mathcal{S}$	
<b>Given:</b>	A connected graph $G$ and a partial order $R$ on $V(G)$ .
<b>Question:</b>	Is there an $\mathcal{S}$ -ordering of $G$ that extends $R$ ?

This paper focuses on PSOP-MCS restricted to connected chordal graphs.

### 2.3 Clique Graphs

It has long been known that chordal graphs admit a characterization in terms of their clique trees. Precisely, a connected graph  $G$  is chordal if and only if there exists a tree  $T$  whose vertices one-to-one correspond to maximal cliques of  $G$  so that for every vertex  $v \in V(G)$  the vertices of  $T$  corresponding to all maximal  $v$ -cliques of  $G$  induce a subtree of  $T$  [4, 7, 15, 29]. Such a tree  $T$  is referred to as a *clique tree* of  $G$  [7, 15, 30]. It is a folklore that every chordal graph  $G$  has at most  $|V(G)|$  maximal cliques [13], and hence every clique tree of  $G$  contains at most  $|V(G)|$  vertices.

Another relevant notion is *clique graph*, first introduced by Galinier, Habib, and Paul [14]. Precisely, the clique graph of a connected chordal graph  $G$ , denoted  $C(G)$ , is the graph whose vertex set is exactly the set of all maximal cliques of  $G$ , and two vertices  $K$  and  $K'$  in



■ **Figure 1** A connected chordal graph (left) and its clique graph (right). In the clique graph, all omitted edge weights are 1.

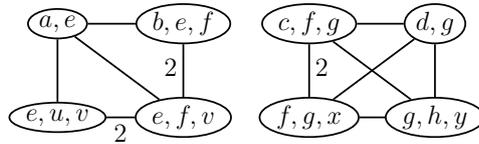
the clique graph are adjacent if and only if  $K \cap K'$  is a minimal  $u$ - $v$  separator of  $G$  for all  $u \in K \setminus K'$  and all  $v \in K' \setminus K$ . Each edge  $KK'$  in the clique graph  $C(G)$  is associated with the label  $K \cap K'$  and with the weight  $|K \cap K'|$ .

For clarity, hereinafter we call vertices in a clique tree or a clique graph *nodes*.

► **Lemma 1** ([14]). *Let  $G$  be a connected chordal graph  $G$ , and let  $K$  and  $K'$  be two maximal cliques in  $G$ . Then,  $K$  and  $K'$  are adjacent in  $C(G)$  if and only if they are adjacent in some clique tree of  $G$ .*

In effect, Lemma 1 asserts that the clique graph of a connected chordal graph is the union of all clique trees of the same graph.

For a label  $S$  of some edge in  $C(G)$ , we use  $C(G) \ominus S$  to denote the graph obtained from  $C(G)$  by deleting all edges with the label  $S$ . For example, for the chordal graph  $G$  in Figure 1,  $C(G) \ominus \{f\}$  is the graph shown in Figure 2.



■ **Figure 2**  $C(G) \ominus \{f\}$  where  $G$  is the connected chordal graph in Figure 1.

► **Lemma 2** ([24]). *Let  $G$  be a connected chordal graph, and let  $S$  be the label of an edge in  $C(G)$  with the minimum weight. Then,*

- (i) *for every  $v \in V(G) \setminus S$ , all maximal  $v$ -cliques of  $G$  are in the same connected component of  $C(G) \ominus S$ ; and*
- (ii) *for every  $u, v \in V(G) \setminus S$ ,  $S$  is a  $u$ - $v$  separator in  $G$  if and only if any maximal  $u$ -clique and any maximal  $v$ -clique of  $G$  are in different connected components of  $C(G) \ominus S$ .*

## 2.4 Graph Search Paradigms

Now we give definitions of three graph search paradigms, namely, MCS, generic search, and Prim search. Each search paradigm regulates how the vertices of a graph are visited one after another without repeating. Our focus is PSOP-MCS, but our algorithm for PSOP-MCS resorts to Prim search of clique graphs and generic search of layer structures (defined in Section 3) of clique graphs.

Both MCS and generic search arbitrarily select one vertex to visit first. Then, MCS picks an unvisited vertex having the maximum number of visited neighbors as the next to visit [29], while the generic search picks an unvisited vertex having at least one visited neighbor as the next to visit [28].

Prim's algorithm is one of the most famous algorithms for finding minimum spanning trees [12, 19, 23]. In a nutshell, starting from a tree consisting of an edge with the minimum weight, the algorithm grows the tree by adding edges, one-by-one, of minimum possible weights without destroying the tree structure, until the tree becomes a spanning tree. By turning "minimum" to "maximum" in the algorithm, it instead returns a maximum spanning tree. Algorithm 1 delineates the Prim search modified from Prim's algorithm [24]. Generally speaking, starting from any arbitrary vertex, it picks as the next one to be visited a so far unvisited vertex incident to an edge with the maximum weight among all edges between visited vertices and unvisited vertices.

■ **Algorithm 1** Prim Search.

---

**Input:** A connected graph  $G$  of  $n$  vertices where every edge has a weight.  
**Output:** An ordering  $\pi$  of  $V(G)$ .

```

1:  $\pi(v) \leftarrow 1$ , where  $v$  is an arbitrary vertex of  $G$ ;           /* the first visited vertex */
2:  $S \leftarrow \{v\}$ ;
3: for  $i = 2, 3, \dots, n$  do
4:   let  $v$  be a vertex in  $V(G) \setminus S$  incident to an edge with the maximum weight among
      all edges between  $S$  and  $V(G) \setminus S$ ;
5:    $\pi(v) \leftarrow i$ ;                                           /* visit  $v$  */
6:    $S \leftarrow S \cup \{v\}$ ;
7: return  $\pi$ ;
```

---

Following [24], we call an ordering obtained from applying Algorithm 1 to a graph  $G$  a *Prim ordering* of  $G$ . Prim orderings of the clique graph of a chordal graph have an appealing property in respect of their clique graphs, as stated in the following lemma.

► **Lemma 3** ([24]). *Let  $G$  be a connected chordal graph, and let  $(K_1, K_2, \dots, K_t)$  be a Prim ordering of  $C(G)$ . For every  $i \in [t]$ , the subgraph of  $C(G)$  induced by  $\{K_1, K_2, \dots, K_i\}$  is the clique graph of the subgraph of  $G$  induced by  $\bigcup_{j \in [i]} K_j$ .*

Let  $\sigma$  be an ordering of  $V(G)$ , and let  $\pi$  be an ordering of the maximal cliques of  $G$ . For  $v \in V(G)$ , we use  $K_\pi^v$  to denote the first  $v$ -clique in  $\pi$ . We say that  $\sigma$  is a *generation* of  $\pi$  (or  $\pi$  *generates*  $\sigma$ ) if for all  $x, y \in V(G)$  it holds that  $K_\pi^x <_\pi K_\pi^y$  implies  $x <_\sigma y$ . Precisely, for an ordering  $\pi = (K_1, K_2, \dots, K_t)$  of maximal cliques of  $G$  and  $i \in [t-1]$ , let  $V(\pi, i) = \bigcup_{j \in [i]} K_j$  be the set of vertices of  $G$  contained in at least one of the first  $i$  cliques from  $\pi$ . For  $i \in [t] \setminus \{1\}$ , let  $K(\pi, i) = K_i \setminus V(\pi, i-1)$  be the set of vertices of  $G$  contained in  $K_i$  but not in any other cliques before  $K_i$  in  $\pi$ . Besides, let  $K(\pi, 1) = K_1$ . Then,  $\sigma$  is a generation of  $\pi$  if and only if it is of the form  $(\overrightarrow{K(\pi, 1)}, \overrightarrow{K(\pi, 2)}, \dots, \overrightarrow{K(\pi, t)})$ , where for a set  $X$ ,  $\overrightarrow{X}$  can be any ordering of  $X$ .

► **Lemma 4** ([24]). *Let  $G$  be a connected chordal graph. Then, an ordering of  $V(G)$  is an MCS ordering of  $G$  if and only if it is a generation of some Prim ordering of  $C(G)$ .*

### 3 Layer Structures of Clique Graphs

In this section, we introduce the notion of layer structure, and explore a number of structural properties pertinent to this notion. Throughout this section, let  $G$  be a connected chordal graph, and let  $K^*$  be a maximal clique of  $G$ .

Let  $\mathcal{U}$  be the set of connected components of  $C(G)$  after the deletion of all edges with the minimum weight. For the sake of readability, let us call each  $U \in \mathcal{U}$  a *unit*. If a maximal clique  $K$  of  $G$  is a node in a unit, we say that  $K$  is *contained* in this unit. We use  $U^K$  to

denote the unit containing  $K$ . Besides, we use  $\mathcal{K}(U)$  to denote the set of maximal cliques of  $G$  contained in a unit  $U$ , and use  $\mathcal{V}(U)$  to denote the set of vertices of  $G$  contained in nodes of  $U$ , i.e.,  $\mathcal{V}(U) = \bigcup_{K \in \mathcal{K}(U)} K$ . We say that an edge in  $C(G)$  *crosses* two units if the two endpoints of the edge are respectively from the two units.

The layer structure we shall study is a refinement of the clique graph  $C(G)$ . The following two lemmas pinpoint where the refinement lies.

► **Lemma 5.** *The weights of edges of  $C(G)$  whose both endpoints are contained in the same unit are greater than the minimum weight of edges of  $C(G)$ .*

**Proof.** Let  $KK'$  be an edge with the minimum weight in  $C(G)$ , and let  $S = K \cap K'$ . Therefore,  $S$  is a  $u$ - $v$  separator for all  $u \in K \setminus S$  and  $v \in K' \setminus S$ . Then, by Lemma 2 (ii),  $K$  and  $K'$  are in different connected components of  $C(G) \ominus S$ , implying that  $K$  and  $K'$  are contained in different units. ◀

Lemma 5 equivalently asserts that every unit  $U$  is exactly the subgraph of  $C(G)$  induced by  $\mathcal{K}(U)$ . To put it another way, an edge in  $C(G)$  crosses two units if and only if it has the minimum weight in  $C(G)$ .

► **Lemma 6.** *Let  $U$  and  $U'$  be two units from  $\mathcal{U}$  so that there are edges in  $C(G)$  crossing  $U$  and  $U'$ . Then, all edges in  $C(G)$  crossing  $U$  and  $U'$  have the same label.*

**Proof.** Towards a contradiction, assume that  $C(G)$  contains two distinct edges  $K_1K_2$  and  $K_3K_4$  crossing  $U$  and  $U'$  with different labels. Let  $S = K_1 \cap K_2$  and let  $S' = K_3 \cap K_4$ . So,  $S \neq S'$ . Obviously,  $U$  and  $U'$  both remain connected in  $C(G) \ominus S$ . Moreover, since the edge  $K_3K_4$  is present in  $C(G) \ominus S$ ,  $U$  and  $U'$  are in the same connected component of  $C(G) \ominus S$ . This means that  $K_1$  and  $K_2$  are in the same connected component of  $C(G) \ominus S$ . However, this contradicts Lemma 2 (ii). ◀

Now we are ready to define the layer structure.

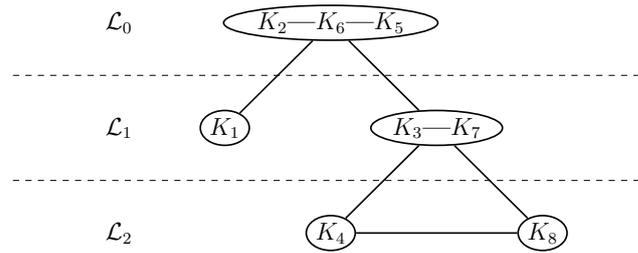
► **Definition 7 (Layer Structure).** *The layer structure of  $C(G)$  rooted by  $K^*$  is a graph with the vertex set  $\mathcal{U}$  so that there is an edge between two units in  $\mathcal{U}$  if and only if there exists at least one edge in  $C(G)$  crossing the two units. The label and the weight of an edge  $UU'$  in the layer structure are respectively  $K \cap K'$  and  $|K \cap K'|$ , where  $KK'$  can be any edge in  $C(G)$  crossing  $U$  and  $U'$ . The unit  $U^{K^*}$  is called the root of the layer structure. A unit is in the  $i$ -th layer if it is at a distance  $i$  from the root, where the distance between two units is defined as the length of a shortest path between them in the layer structure.*

See Figure 3 for an illustration of Definition 7. By Lemma 6, the labels and the weights of edges in the layer structure are well-defined. Let  $\mathcal{L}_i$  be the set of all units in the  $i$ -th layer, and let  $\mathcal{L}_{\leq i} = \bigcup_{j \in [i] \cup \{0\}} \mathcal{L}_j$ . Obviously,  $\mathcal{L}_0 = \{U^{K^*}\}$ . In addition, if two units from respectively two layers  $\mathcal{L}_i$  and  $\mathcal{L}_j$  are adjacent, it holds that  $|i - j| \leq 1$ . Recall that  $G$  is connected. Hence,  $C(G)$  and the layer structure are connected too. As a result, every unit in  $\mathcal{U}$  is in some layer.

In the following, we explore numerous properties of the layer structure.

► **Property 1.** *Let  $UU'$  be an edge in the layer structure. Then, every path between  $U$  and  $U'$  in the layer structure contains an edge with the same label as  $UU'$ .*

**Proof.** Let  $KK'$  be an edge in  $C(G)$  crossing the two units  $U$  and  $U'$ . Let  $S = K \cap K'$ . It is clear that  $S$  is both the label of  $KK'$  and the label of  $UU'$ . By Lemma 2 (ii),  $K$  and  $K'$  are disconnected in  $C(G) \ominus S$ . It follows that every  $K$ - $K'$  path in  $C(G)$  contains at least one edge with the label  $S$ . Then, by Lemma 5, we know that every  $U$ - $U'$  path in the layer structure contains at least one edge with the label  $S$ . ◀



■ **Figure 3** The layer structure of the chordal graph in Figure 1 rooted by  $K_2$  (or  $K_5, K_6$ ).

► **Property 2.** Let  $U_1U_2$  and  $U_3U_4$  be two distinct edges of the layer structure with the same label  $S$ . Then, the units in the set  $\{U_1, U_2, U_3, U_4\}$  are pairwise adjacent in the layer structure, and all edges among them have the same label  $S$ .

**Proof.** Let  $K_1K_2$  and  $K_3K_4$  be two distinct edges of  $C(G)$  both with the label  $S$ , where  $K_i \in \mathcal{K}(U_i)$  for every  $i \in [4]$ . We first show that  $U_1, U_2, U_3$ , and  $U_4$  are pairwise adjacent in the layer structure. Assume, for the sake of contradiction, that one from  $\{U_1, U_2\}$  is not adjacent to one from  $\{U_3, U_4\}$ . By symmetry, suppose that  $U_1 \neq U_3$ , and  $U_1$  is not adjacent to  $U_3$  in the layer structure.

We claim that  $K_1$  and  $K_3$  are disconnected in  $C(G) \ominus S$ . Assume for contradiction that  $K_1$  and  $K_3$  are connected in  $C(G) \ominus S$ . Let  $C'$  be the connected component of  $C(G) \ominus S$  containing  $K_1$  and  $K_3$ , and let  $V'$  be the set of vertices of  $G$  contained in nodes of  $C'$ . By the minimality of  $|S|$ , there exists a Prim ordering of  $C(G)$  so that all maximal cliques of  $G$  contained in  $C'$  are visited before all the other maximal cliques of  $G$ . Additionally, in light of Lemma 2 (ii), the subgraph of  $C(G)$  induced by the nodes in  $C'$  does not contain any edge with the label  $S$ . Then, by Lemma 3,  $C'$  is the clique graph of  $G[V']$ . Obviously,  $S \subsetneq V'$ . By Lemma 1,  $C'$  contains all clique trees of  $G[V']$  as subgraphs. We fix a clique tree of  $G[V']$ . There is a unique  $K_1$ - $K_3$  path in the clique tree, and by the definition of clique trees, all nodes on this path are  $S$ -cliques. Obviously, this path is also present in  $C(G) \ominus S$ . The label of each edge on this path cannot be  $S$ , since such edges are absent in  $C(G) \ominus S$ . So, the labels of all edges on this path properly contain  $S$ . This indicates that  $K_1$  and  $K_3$  are contained in the same unit in the layer structure. However, this contradicts that  $U_1 \neq U_3$ .

So, it holds that  $K_1$  and  $K_3$  are disconnected in  $C(G) \ominus S$ . Then, according to Lemma 2 (ii),  $S$  is a  $u$ - $v$  separator for all  $u \in K_1 \setminus S$  and all  $v \in K_3 \setminus S$ . By the minimality of  $|S|$  and the definition of clique graphs,  $K_1K_3$  is an edge of  $C(G)$  with the label  $S$ . As a result,  $U_1U_3$  is an edge of the layer structure, contradicting that  $U_1$  and  $U_3$  are nonadjacent in the layer structure.

Now we can conclude that the units in  $\{U_1, U_2, U_3, U_4\}$  are pairwise adjacent in the layer structure. Then, from Property 1 and the fact that both  $U_1U_2$  and  $U_3U_4$  have the label  $S$ , it follows that the labels of the edges among  $U_1, U_2, U_3$ , and  $U_4$  are all  $S$ . ◀

Note that in Property 2 it may be that  $\{U_1, U_2\} \cap \{U_3, U_4\} \neq \emptyset$ .

► **Property 3.** Every unit  $U$  in the  $i$ -th layer  $\mathcal{L}_i$  where  $i \geq 1$  is adjacent to exactly one unit from the layer  $\mathcal{L}_{i-1}$ .

**Proof.** Let  $U$  be a unit from the  $i$ -th layer. The statement holds trivially for  $i = 1$ . Consider now the case where  $i \geq 2$ . Assume, for the sake of contradiction, that there are two distinct units  $U_1$  and  $U_2$  in the layer  $\mathcal{L}_{i-1}$  both adjacent to  $U$  in the layer structure. Let  $S_1$  and  $S_2$  be the labels of  $UU_1$  and  $UU_2$ , respectively. By Definition 7, there is a path from the root

to  $U_1$ , and a path from the root to  $U_2$  in the layer structure. Then, as  $i \geq 2$ , there exists a  $U_1$ - $U_2$  path  $P$  of length at least two in the layer structure such that all inner units of  $P$  are from  $\mathcal{L}_{\leq i-2}$ . Our proof proceeds by distinguishing between the following two cases.

**Case 1:**  $S_1 = S_2$ .

By Property 2,  $U_1U_2$  is an edge with the label  $S_1$  in the layer structure. By Property 1, there exists an edge in  $P$  with the label  $S_1$ . By Property 2,  $U$  is adjacent to the two endpoints of this edge, which is impossible since  $U$  is from the  $i$ -th layer but at least one of the two endpoints of the edge is from  $\mathcal{L}_{\leq i-2}$ .

**Case 2:**  $S_1 \neq S_2$ .

By Property 1, there exists an edge in  $E(P) \cup \{UU_2\}$  with the label  $S_1$ . As  $S_1 \neq S_2$ , this edge belongs to  $P$ . By Property 2,  $U$  is adjacent to the two endpoints of this edge. However, analogous to the discussion in Case 1, we know that this is impossible.

As both cases lead to some contradictions, we know that  $U$  is adjacent to exactly one unit from  $\mathcal{L}_{i-1}$ . ◀

Now for each unit  $U$  from a layer  $\mathcal{L}_i$  where  $i \geq 1$ , we call the only unit from the layer  $\mathcal{L}_{i-1}$  adjacent to  $U$  the *parent* of  $U$ , and use  $\text{parent}(U)$  to denote it. Correspondingly, we say that  $U$  is a *child* of  $\text{parent}(U)$ . Furthermore, for a unit  $U$  from a layer  $\mathcal{L}_i$  and a unit  $U'$  from a layer  $\mathcal{L}_j$  such that  $i < j$ , we say that  $U'$  is a *descendant* of  $U$  if there is a path from the root to  $U'$  through  $U$  (i.e.,  $U$  is also on the path) in the layer structure. For a unit  $U$ , let  $\text{dst}(U)$  be the set of all descendants of  $U$ , and let  $\text{dst}[U] = \text{dst}(U) \cup \{U\}$ .

► **Property 4.** Let  $U_1$  and  $U_2$  be two units in the same layer  $\mathcal{L}_i$  where  $i \geq 1$ . Then, the following statements are equivalent:

- (1)  $U_1$  and  $U_2$  are adjacent.
- (2)  $\text{parent}(U_1) = \text{parent}(U_2)$ , and the label of the edge between  $U_1$  and its parent equals that between  $U_2$  and its parent.

**Proof.** From Property 2, we know that Statement (2) implies Statement (1). In the following, we show that Statement (1) implies Statement (2).

Assume that  $U_1$  and  $U_2$  are adjacent, and let  $S$  be the label of the edge between them. Let  $U_3 = \text{parent}(U_1)$  and let  $U_4 = \text{parent}(U_2)$ . We first prove that  $U_1$  and  $U_2$  have the same parent, i.e.,  $U_3 = U_4$ . Assume, for contradiction, that  $U_3 \neq U_4$ . By Properties 2 and 3, this implies that neither the label of  $U_1U_3$  nor the label of  $U_2U_4$  is  $S$ . In addition, it also holds that  $i > 1$ , implying that there exists a  $U_3$ - $U_4$  path  $P$  of length at least two in the layer structure whose inner units are all from  $\mathcal{L}_{\leq i-2}$  (cf. the proof of Property 3). Then, by Property 1, there exists an edge in  $P$  with the label  $S$ . By Property 2, the two endpoints of this edge are adjacent to  $U_1$  in the layer structure. However, this is impossible since at least one endpoint of this edge is from  $\mathcal{L}_{\leq i-2}$  but  $U_1$  is from the layer  $\mathcal{L}_i$ . This completes the proof for that  $U_3 = U_4$ . Having  $U_3 = U_4$ , from Properties 1 and 2, it follows that the labels of  $U_1U_3$  and  $U_2U_4$  are both  $S$ . ◀

Property 4 implies that if a subset of units in the same layer  $\mathcal{L}_i$  are connected in the layer structure restricted to  $\mathcal{L}_i$ , then they are pairwise adjacent. For ease of exposition, we group units in the same layer into *bags* so that two units are in the same bag if they are adjacent. By Property 3, if we ignore edges inside all bags in the layer structure, we obtain a tree rooted at  $U^{K^*}$ . As an important consequence, every path connecting two units from the same layer  $\mathcal{L}_i$  is completely contained in  $\mathcal{L}_{\leq i}$ .

In addition, Property 3 indicates that every nonroot unit  $U$  in a layer  $\mathcal{L}_i$  is adjacent to exactly one unit in  $\mathcal{L}_{\leq i-1}$ , and this unit is its parent in  $\mathcal{L}_{i-1}$ . Property 4 further strengthens that  $\text{parent}(U)$  indeed separates all descendants of  $\text{parent}(U)$  from all the other units. In other

words,  $\{\text{parent}(U)\}$  is a  $U'$ - $U''$  separator in the layer structure for all  $U' \in \text{dst}(\text{parent}(U))$  and all  $U'' \in \mathcal{U} \setminus \text{dst}[\text{parent}(U)]$ . An important consequence is that every generic search ordering of the layer structure starting from the root visits every nonroot unit after (not necessarily consecutive) its parent.

The next property identifies labels of edges between adjacent units.

► **Property 5.** *The label of every edge  $UU'$  in the layer structure is  $\mathcal{V}(U) \cap \mathcal{V}(U')$ .*

**Proof.** Let  $U$  and  $U'$  be two adjacent units in the layer structure. Let  $S = \mathcal{V}(U) \cap \mathcal{V}(U')$ , and let  $S'$  be the label of the edge between  $U$  and  $U'$  in the layer structure. Clearly,  $S' \subseteq S$ . Therefore, to complete the proof, it needs only to show that every  $v \in S$  is contained in  $S'$ . In light of Properties 3 and 4, only the two cases described below may occur.

**Case 1:** one of  $U$  and  $U'$  is the parent of the other.

For the sake of contradiction, assume that there exists  $v \in S \setminus S'$ . By Lemma 6, all edges between  $\mathcal{K}(U)$  and  $\mathcal{K}(U')$  in  $C(G)$  have the same label  $S'$ . By the definition of the layer structure,  $|S'|$  is a minimum edge weight in  $C(G)$ . Then, by Properties 3 and 4,  $U$  and  $U'$  are disconnected in the layer structure after removing all edges with the label  $S'$ . This is equivalent to every  $K \in \mathcal{K}(U)$  and every  $K' \in \mathcal{K}(U')$  being disconnected in  $C(G) \ominus S'$ . However, as  $v \in \mathcal{V}(U) \cap \mathcal{V}(U')$ , this violates Lemma 2 (i).

**Case 2:**  $U$  and  $U'$  are in the same bag.

By Property 4,  $U$  and  $U'$  have the same parent, say  $\hat{U}$ , and the labels of edges among units in the same bag as  $U$  and their parent  $\hat{U}$  are  $S'$ . Then, similar to Case 1, it is easy to see that  $U$  and  $U'$  are disconnected in the layer structure after removing all edges with the label  $S'$ , which violates Lemma 2 (i) too.

As both cases violate Lemma 2, we know that Property 5 holds. ◀

► **Property 6.** *Let  $v$  be a vertex in  $G$ . Let  $i$  be the minimum integer such that  $\mathcal{L}_i$  contains a maximal  $v$ -clique. Then, all maximal  $v$ -cliques of  $G$  contained in  $\mathcal{L}_i$  are in one unit.*

**Proof.** Let  $v$  and  $i$  be as stipulated in the statement of Property 6. Assume for contradiction that there exist two maximal  $v$ -cliques  $K$  and  $K'$  respectively from two different units  $U$  and  $U'$  in the layer  $\mathcal{L}_i$ . Our proof is completed by considering the following two cases.

**Case 1:**  $U$  and  $U'$  are adjacent in the layer structure.

By Property 5,  $v$  is in the label of the edge  $UU'$ . By Property 4,  $U$  and  $U'$  have the same parent, say,  $\hat{U}$ . Moreover, by Properties 2 and 4, the edges  $U\hat{U}$  and  $U'\hat{U}$  have the same label as the edge  $UU'$ . It follows that  $\hat{U}$  contains some  $v$ -clique. However, this contradicts that  $i$  is the minimum integer with  $\mathcal{L}_i$  containing a maximal  $v$ -clique.

**Case 2:**  $U$  and  $U'$  are not adjacent in the layer structure.

Note that in this case  $i \geq 1$ . As the subgraph of  $C(G)$  induced by the set of maximal  $v$ -cliques is connected, there is a  $K$ - $K'$  path  $P$  in  $C(G)$  so that  $v$  is in the label of every edge on the path. By Properties 3 and 4, this path contains at least one maximal clique of  $G$  contained in  $\text{parent}(U)$ . However, similar to Case 1, this is in contradiction with the definition of  $i$ .

As each of the above two cases leads to a certain contradiction, Property 6 holds. ◀

Property 6 shows that for every vertex  $v$  in  $G$ , there is a unique unit that contains a maximal  $v$ -clique and is at the least distance to the root in the layer structure. Let  $U^v$  be such a unique unit for  $v$ .

► **Property 7.** *For each  $v \in V(G)$ , every unit containing a maximal  $v$ -clique of  $G$  is from  $\text{dst}[U^v]$ .*

## 77:10 Algorithm for MCS Partial Search on Chordal Graphs

**Proof.** Assume, for the sake of contradiction, there exists  $v \in V(G)$  and a unit  $U$  such that  $v \in \mathcal{V}(U)$  and  $U \notin \text{dst}[U^v]$ . By Property 6,  $U$  and  $U^v$  cannot be in the same layer. Then, as  $U \notin \text{dst}[U^v]$ , every  $U^v$ - $U$  path in the layer structure contains the parent of  $U^v$ . As all maximal  $v$ -cliques are connected in  $C(G)$ , and  $v \in \mathcal{V}(U) \cap \mathcal{V}(U^v)$ , we know that  $v \in \mathcal{V}(\text{parent}(U^v))$ . However, this contradicts the definition of  $U^v$ . ◀

► **Property 8.** *Let  $U$  and  $U'$  be two units from the same bag, and let  $\hat{U}$  be their parent. Then, for every edge  $KK'$  in  $C(G)$  so that  $K \in \mathcal{K}(U)$  and  $K' \in \mathcal{K}(U')$ , there exists  $\hat{K} \in \mathcal{K}(\hat{U})$  which is adjacent to both  $K$  and  $K'$  in  $C(G)$ .*

**Proof.** Let  $K \in \mathcal{K}(U)$  and  $K' \in \mathcal{K}(U')$  be as stipulated in Property 8. Let  $S = K \cap K'$ . As  $U$  and  $U'$  are in the same bag, they are adjacent in the layer structure. By Lemma 6 and Definition 7, the label of the edge  $UU'$  is  $S$ . This means that  $|S|$  is the minimum weight of edges in  $C(G)$ . By Properties 2 and 4, the labels of the edges  $U\hat{U}$  and  $U'\hat{U}$  are also  $S$ . As a result, there exists  $\hat{K} \in \mathcal{K}(\hat{U})$  so that  $S \subseteq \hat{K}$ . By Properties 3 and 4,  $\hat{K}$  is disconnected from  $K$  in  $C(G) \ominus S$ . Then, according to Lemma 2 (ii) and the minimality of  $|S|$ ,  $S$  is a minimal  $u$ - $v$  separator in  $G$  for all  $u \in K \setminus \hat{K}$  and all  $v \in \hat{K} \setminus K$ . Therefore,  $\hat{K}$  and  $K$  are adjacent in  $C(G)$ . Similarly,  $\hat{K}$  is also adjacent to  $K'$  in  $C(G)$ . ◀

Now we study two lemmas which provide insight into connections among MCS orderings of  $G$ , Prim orderings of  $C(G)$ , and generic search orderings of the layer structure. We say that a Prim ordering  $\pi$  of the clique graph  $C(G)$  *respects* a partial order  $R$  on  $V(G)$  if for every  $(x, y) \in R$  it holds that  $K_\pi^x <_\pi K_\pi^y$  or  $K_\pi^x = K_\pi^y$ . By saying that an ordering starts with elements in a subset, we mean the elements in the subset are before all the other elements in the ordering.

► **Lemma 8.** *Let  $R$  be a partial order on  $V(G)$ . There is an MCS ordering of  $G$  extending  $R$  if and only if there is a Prim ordering of  $C(G)$  respecting  $R$ . Moreover, given a Prim ordering of  $C(G)$  which starts with some node  $K$  and respects  $R$ , we can construct an MCS ordering of  $G$  starting with the vertices in  $K$  and extending  $R$  in polynomial time.*

**Proof.** We start the proof with the first statement. For the forward direction, let  $\sigma$  be an MCS ordering of  $G$  extending  $R$ . By Lemma 4, there exists a Prim ordering  $\pi$  of  $C(G)$  that generates  $\sigma$ . For every  $(x, y) \in R$ , since  $\sigma$  extends  $R$ , it holds that  $x <_\sigma y$ . As  $\pi$  generates  $\sigma$ , it holds that  $K_\pi^x <_\pi K_\pi^y$  or  $K_\pi^x = K_\pi^y$ . For the backward direction, let  $\pi = (K_1, K_2, \dots, K_t)$  be a Prim ordering of  $C(G)$  so that for all  $(x, y) \in R$  it holds that either  $K_\pi^x <_\pi K_\pi^y$  or  $K_\pi^x = K_\pi^y$ . Let  $\sigma$  be a generation of  $\pi$  so that for every  $i \in [t]$  it holds that  $\sigma$  restricted to  $K(\pi, i)$  extends  $R$  restricted to  $K(\pi, i)$ , i.e.,  $\sigma|_{K(\pi, i)}$  is a linear extension of  $R|_{K(\pi, i)}$ . As for every  $i, j \in [t]$  such that  $i \neq j$ ,  $K(\pi, i)$  and  $K(\pi, j)$  are disjoint,  $\sigma$  is well-defined. By Lemma 4,  $\sigma$  is an MCS ordering of  $G$ . To complete the proof for the first statement, it suffices to show that  $\sigma$  is a linear extension of  $R$ . Let  $(x, y) \in R$ . If  $K_\pi^x = K_\pi^y$ , i.e.,  $x$  and  $y$  are contained in some  $K(\pi, i)$  where  $i \in [t]$ , as  $\sigma|_{K(\pi, i)}$  extends  $R|_{K(\pi, i)}$ , it holds that  $x <_\sigma y$ . Otherwise,  $K_\pi^x <_\pi K_\pi^y$  holds. Then, as  $\sigma$  is a generation of  $\pi$ ,  $x <_\sigma y$  holds too. This completes the proof that  $\sigma$  extends  $R$ .

Concerning the second statement, observe that the above proof for the backward direction is constructive, and the polynomial-time solvability follows from the fact that computing a linear extension of a partial order can be done in polynomial time. ◀

By Property 6, each partial order  $R$  on  $V(G)$  specifies a partial order on the units:  $\mathcal{Q}^R = \{(U^x, U^y) \mid (x, y) \in R\}$ .

► **Lemma 9.** *Let  $R$  be a partial order on  $V(G)$ . Then, if there is an MCS ordering of  $G$  that starts with vertices from  $K^*$  and extends  $R$ , there is a generic search ordering of the layer structure rooted by  $K^*$  that starts with  $U^{K^*}$  and extends  $\mathcal{Q}^R$ .*

**Proof.** Let  $\sigma$  be an MCS ordering of  $G$  that starts with the vertices from  $K^*$  and extends  $R$ . By Lemma 4, there is a Prim ordering  $\pi$  of  $C(G)$  that generates  $\sigma$ . Obviously,  $K^*$  is the first node in  $\pi$ . Then, by Algorithm 1, Lemma 5, and Definition 7, we know that for every unit  $U$ , the nodes in  $U$  are consecutive in  $\pi$ . Moreover, in view of Properties 3 and 4, for every nonroot unit  $U$ , all nodes from  $\text{parent}(U)$  are before all nodes from  $U$  in  $\pi$ . Consequently, there is a linear order  $\pi' = (U_1, U_2, \dots, U_t)$  of the units in the layer structure so that

- (1)  $U_1 = U^{K^*}$ ;
- (2) for every nonroot unit  $U$ ,  $\text{parent}(U)$  is before  $U$  in  $\pi'$ ; and
- (3) for every  $i \in [t]$ , all nodes of  $U_i$  are consecutive in the Prim ordering  $\pi$ .

Conditions (1) and (2) mean that  $\pi'$  is a generic search ordering of the layer structure starting with the root. It remains to show that  $\pi'$  extends  $\mathcal{Q}^R$ . For this purpose, let  $(x, y) \in R$  such that  $U^x \neq U^y$ . As  $\sigma$  extends  $R$ ,  $x$  is before  $y$  in  $\sigma$ . As  $\pi$  generates  $\sigma$ , either  $K_\pi^x = K_\pi^y$  holds or  $K_\pi^x <_\pi K_\pi^y$  holds. Condition (2) given above and Property 7 imply that  $K_\pi^v$  is contained in  $U^v$  for all  $v \in V(G)$ . Then, as  $U^x \neq U^y$ , it must be that  $K_\pi^x <_\pi K_\pi^y$ . Finally, by Condition (3) given above, we know that  $U^x$  is before  $U^y$  in  $\pi'$ . This completes the proof that  $\pi'$  extends  $\mathcal{Q}^R$ . ◀

## 4 A Dynamic Programming Algorithm for PSOP

In this section, we present a polynomial-time dynamic programming algorithm for PSOP-MCS restricted to chordal graphs.

For an ordering  $\pi$  of units in a layer structure, and an ordering  $\sigma$  of nodes contained in a unit  $U$ , *realizing*  $\pi$  by  $\sigma$  is the operation of replacing  $U$  in  $\pi$  with  $\sigma$ . For instance, for  $\pi = (U_1, U_2, U_3, U_4)$  and  $\sigma = (K_1, K_2, K_3)$  where  $\{K_1, K_2, K_3\}$  is the set of nodes contained in  $U_2$ , realizing  $\pi$  by  $\sigma$  results in the linear order  $(U_1, K_1, K_2, K_3, U_3, U_4)$ .

► **Theorem 10.** *PSOP-MCS restricted to chordal graphs is polynomial-time solvable.*

**Proof.** Let  $I = (G, R)$  be an instance of PSOP-MCS where  $G$  is a connected chordal graph of  $n$  vertices. Our algorithm consists of the following steps.

**Step 1** We sort the weights of edges in the clique graph  $C(G)$  of  $G$  in increasing order. Let  $(w(1), w(2), \dots, w(t))$  be this order, where  $t$  is the number of different weights of edges in  $C(G)$ . Hence  $w(1) < w(2) < \dots < w(t)$  holds. Notice that  $t = O(n)$  since edges of  $C(G)$  may have at most  $n - 2$  different weights.

**Step 2** For each  $i \in [t]$ , let  $C^i(G)$  be the graph obtained from  $C(G)$  by removing edges whose weights are from  $\{w(1), w(2), \dots, w(i)\}$ . Let  $C^0(G) = C(G)$ . Clearly, every  $C^i(G)$ , where  $i \in [t]$ , is obtained from  $C^{i-1}(G)$  by deleting all edges with the minimum weight.

**Step 3** We maintain a binary dynamic programming table  $D(i, H, K)$ , where  $i \in [t] \cup \{0\}$ ,  $H$  is a connected component of  $C^i(G)$ , and  $K$  is a node in  $H$ . (Note that for  $i = 0$ , we have that  $H = C(G)$ .) As  $t = O(n)$ , every chordal graph of  $n$  vertices has at most  $n$  maximal cliques, and  $K$  is a node in  $H$ , the table has  $O(n^2)$  entries.

For each connected component  $H$  of some  $C^i(G)$ , let  $\mathcal{V}(H)$  be the subset of vertices of  $G$  contained in nodes of  $H$ . We define  $D(i, H, K)$  to be 1 if there is a Prim ordering of  $H$  which starts with  $K$  and respects  $R|_{\mathcal{V}(H)}$ , and define  $D(i, H, K)$  to be 0 otherwise. (We elaborate on how to compute the table later.)

**Step 4** After the table is computed, if  $D(0, C(G), K) = 0$  for all maximal cliques  $K$  of  $G$ , by the definition of the table there is no Prim ordering of  $C(G)$  respecting  $R$ , and by Lemma 8, the given instance  $I$  is a No-instance. Otherwise, there exists a maximal clique  $K$  of  $G$  such that  $D(0, C(G), K) = 1$ . By the definition of the table and by Lemma 8, there is an MCS ordering of  $G$  extending  $R$ . Therefore, in this case, we conclude that  $I$  is a Yes-instance.

Computing the table defined above lies at the core of the algorithm, which is the main focus of the remainder of the proof. To this end, we first show that each graph  $H$  used in Step 3 is a clique graph of a connected chordal graph.

▷ **Claim 11.** For every  $i \in [t] \cup \{0\}$ , every connected component  $H$  of  $C^i(G)$  is the clique graph of the subgraph of  $G$  induced by  $\mathcal{V}(H)$ .

Proof of Claim 11. We prove the claim by induction on  $i$ . The statement is clearly true for  $i = 0$ . Now let  $i \in [t]$ , and let  $H$  be a connected component of  $C^i(G)$ . Let  $H'$  be the connected component of  $C^{i-1}(G)$  containing  $H$ . By induction,  $H'$  is the clique graph of  $G[\mathcal{V}(H')]$  which is a connected chordal graph. Obviously,  $H$  is a connected component of  $H'$  after removing all edges with the minimum weight in  $H'$ . As a consequence, there is a Prim ordering of  $H'$  which starts with a node from  $H$  and consecutively visits nodes in  $H$ . Then, by Lemma 3,  $H$  is the clique graph of the subgraph of  $G[\mathcal{V}(H')]$  induced by  $\mathcal{V}(H)$  which is exactly  $G[\mathcal{V}(H)]$ . This completes the proof for the claim. ◁

By Claim 11, each  $H$  in Step 3 is the clique graph of a connected chordal graph. Therefore, all properties and lemmas studied in the previous section apply to  $H$  and each of its layer structures.

Now we show how to compute the table defined in Step 3. We fill the entries  $D(i, H, K)$  in a decreasing order of the values of  $i$ .

- filling the base entries

As  $C^t(G)$  consists of pairwise nonadjacent nodes corresponding to maximal cliques of  $G$ , by the definition of the table, we set  $D(t, H, K) = 1$  for all base entries.

- updating the table

Now we delineate how to update other entries  $D(i, H, K)$ , assuming all entries  $D(i + 1, H', K')$  have been computed. To compute the entry  $D(i, H, K)$ , we first compute the layer structure of  $H$  rooted by  $K$ . Let  $\mathcal{LS}$  be the layer structure. By Definition 7,  $\mathcal{LS}$  can be computed in polynomial time. Note that each unit in  $\mathcal{LS}$  is a connected component of  $C^{i+1}(G)$ . For each vertex  $x$  in  $\mathcal{V}(H)$ , let  $U^x$  be the unit in  $\mathcal{LS}$  which contains  $x$  and is at the least distance from the root of  $\mathcal{LS}$ . By Property 6, such a unit is unique. Besides, for each node  $K'$  of  $H$ , let  $U^{K'}$  be the unit in  $\mathcal{LS}$  containing  $K'$ . Let  $R' = R|_{\mathcal{V}(H)}$  be  $R$  restricted to  $\mathcal{V}(H)$ . Let  $\mathcal{Q}^{R'} = \{(U^x, U^y) \mid (x, y) \in R'\}$ . Now we determine if there is a generic search ordering of  $\mathcal{LS}$  which starts with the root and extends  $\mathcal{Q}^{R'}$ . This can be done in polynomial time [26, Theorem 6]<sup>1</sup>. If this is not the case, by Claim 11 and Lemma 9, there is no MCS ordering of  $G[\mathcal{V}(H)]$  that starts with vertices from  $K$  and extends  $R'$ , and by Claim 11 and Lemma 8 there is no Prim ordering of  $H$  which starts with  $K$  and respects  $R'$ . So in this case, we set  $D(i, H, K) = 0$ . Otherwise, let  $\pi = (U_0, U_1, \dots, U_p)$  be a generic search ordering of  $\mathcal{LS}$  extending  $\mathcal{Q}^{R'}$  so that  $U_0 = U^K$ . Then, we let  $D(i, H, K) = 1$  if and only if

<sup>1</sup> Scheffler [26] showed that the rooted version of the PSOP for the generic search can be solved in polynomial time. In this version, we are given a graph  $G$ , a partial order on  $V(G)$ , and a vertex  $v \in V(G)$ , and the question is whether  $G$  admits a generic search ordering which starts with  $v$  and extends the partial order.

- (1)  $D(i+1, U^K, K) = 1$ , and
- (2) for every  $U_j$ ,  $j \in [p]$ , there exists a node  $K_j$  contained in  $U_j$  such that in  $H$  the node  $K_j$  is adjacent to at least one node from the parent of  $U_j$  in  $\mathcal{LS}$  and, moreover,  $D(i+1, U_j, K_j) = 1$ .

We show the correctness of this step as follows. Observe that in every generic search ordering of  $\mathcal{LS}$  starting from the root, every unit is visited before all its children.

We first prove the “if” direction. Assuming Conditions (1) and (2), let  $\pi'$  be the ordering obtained from  $\pi$  by

- (a) realizing the first unit  $U^K$  by a Prim ordering of  $U^K$  which starts with  $K$  and respects  $R'$  restricted to  $U^K$  (guaranteed by  $D(i+1, U^K, K) = 1$ ), and
- (b) realizing every  $U_j$ , where  $j \in [p]$ , by a Prim ordering of  $U_j$  which starts with  $K_j$  and respects  $R'$  restricted to  $U_j$  (guaranteed by  $D(i+1, U_j, K_j) = 1$ ).

The remainder of the proof for the “if” direction comprises the two claims below.

▷ **Claim 12.**  $\pi'$  is a Prim ordering of  $H$  with the first node being  $K$ .

*Proof of Claim 12.* By Condition (1) and Operation (a), we know that the first node in  $\pi'$  is  $K$ . Besides, from Condition (2) and Operation (b), for every  $U_j$  where  $j \in [p]$  the first node of  $\pi'$  restricted to  $U_j$  is  $K_j$ . By the definition of Prim ordering (Algorithm 1) and the definition of  $\pi'$ , it suffices now to show that for every  $U_j$ ,  $j \in [p]$ , the following condition holds:  $K_j$  is adjacent to at least one node in  $H$  which is before  $K_j$  in  $\pi'$  and is from a different unit adjacent to  $U_j$ . This is the case as by Condition (2),  $K_j$  is adjacent to at least one node from the parent of  $U_j$  in  $\mathcal{LS}$ , and as  $\pi$  is a generic search ordering of  $\mathcal{LS}$  with the root being the first unit, by Properties 3 and 4, the parent of  $U_j$  is before  $U_j$  in  $\pi$ , implying that all nodes in the parent of  $U_j$  are before all nodes of  $U_j$  in  $\pi'$ . ◁

▷ **Claim 13.**  $\pi'$  respects  $R'$ .

*Proof of Claim 13.* To verify that  $\pi'$  respects  $R'$ , let  $(x, y) \in R'$ . Due to Properties 3, 4, 7, and that  $\pi$  is a generic search ordering of the layer structure starting with the root, we know that for every  $v \in V(G)$  the first node in  $\pi'$  containing  $v$  is from  $U^v$ , i.e.,  $K_{\pi'}^v \in \mathcal{K}(U^v)$ . Our proof proceeds by distinguishing between the following two cases. If  $U^x = U^y = U$ , then as  $\pi$  has been realized by a Prim ordering of  $U$  respecting  $R'$  restricted to  $U$  in  $\pi'$ , it holds that  $K_{\pi'}^x <_{\pi'} K_{\pi'}^y$ , or  $K_{\pi'}^x = K_{\pi'}^y$ . Otherwise, as  $\pi$  extends  $\mathcal{Q}^{R'}$ ,  $U^x$  is before  $U^y$  in  $\pi$ . By the definition of  $\pi'$ , maximal  $x$ -cliques in  $U^x$  are before maximal  $y$ -cliques in  $U^y$ . By Properties 3, 4, and 7, none of any units containing a maximal  $y$ -clique is before  $U^y$  in  $\pi$ . Then, from  $K_{\pi'}^x \in \mathcal{K}(U^x)$  and  $K_{\pi'}^y \in \mathcal{K}(U^y)$ , it follows  $K_{\pi'}^x <_{\pi'} K_{\pi'}^y$ . ◁

Now we give the proof for the “only if” direction. To this end, assume that  $D(i, H, K) = 1$ , i.e.,  $H$  admits at least one Prim ordering, say  $\pi'$ , which starts with  $K$  and respects  $R'$ . As  $\pi'$  respects  $R'$ , for each unit  $U$  in the layer structure,  $\pi'$  restricted to  $U$ , i.e.,  $\pi'|_U$ , is a Prim ordering of  $U$  respecting  $R'$  restricted to  $U$ . Consequently,  $D(i+1, U, K') = 1$  where  $K'$  is the first node in  $\pi'|_U$ . This immediately implies that Condition (1) holds. We show below that Condition (2) also holds. Let  $U_j$ ,  $j \in [p]$ , be a unit in  $\mathcal{LS}$ . Let  $K'$  be the first node in  $\pi'|_{U_j}$ . We claim that  $K'$  is adjacent in  $H$  to some node from the parent of  $U_j$  in  $\mathcal{LS}$ . As  $\pi'$  is a Prim ordering of  $H$  and  $K'$  is not the first node in  $\pi'$ ,  $K'$  is adjacent to at least one node, say  $\hat{K}$ , before  $K'$  in  $\pi'$  and, moreover, as  $K'$  is the first node in  $\pi'|_{U_j}$ ,  $\hat{K}$  is from a different unit, say  $\hat{U}$ . If  $\hat{U}$  is the parent of  $U_j$  in  $\mathcal{LS}$ , we are done. Otherwise, by Lemma 5

and Definition 7, we know that nodes in each unit are consecutive in  $\pi'$ . As  $K$  is the first node of  $\pi'$  and  $K$  is contained in the root of  $\mathcal{LS}$ , by Properties 3 and 4, none of the nodes contained in any descendant of  $U_j$  is visited before  $K'$  in  $\pi'$ . It follows that  $\hat{U}$  is in the same bag as  $U_j$ . Then, by Property 8, there is a node from the parent of  $U_j$  adjacent to  $K'$  in  $H$ .

The algorithm runs in polynomial time since the table has at most  $O(n^2)$  entries, and computing the value of each entry can be done in polynomial time as described above. ◀

---

## References

- 1 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. On the end-vertex problem of graph searches. *Discrete Mathematics & Theoretical Computer Science*, 21(1):Nr. 13, 2019. doi:10.23638/DMTCS-21-1-13.
- 2 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. The recognition problem of graph search trees. *SIAM Journal on Discrete Mathematics*, 35(2):1418–1446, 2021. doi:10.1137/20M1313301.
- 3 Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004. doi:10.1007/s00453-004-1084-3.
- 4 Jean R. S. Blair and Barry W. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer, 1993. doi:10.1007/978-1-4613-8369-7\_1.
- 5 Hans L. Bodlaender and Arie M. C. A. Koster. On the maximum cardinality search lower bound for treewidth. *Discrete Applied Mathematics*, 155(11):1348–1372, 2007. doi:10.1016/j.dam.2007.02.004.
- 6 Anna Bretscher, Derek Gordon Corneil, Michel Habib, and Christophe Paul. A simple linear time LexBFS cograph recognition algorithm. *SIAM Journal on Discrete Mathematics*, 22(4):1277–1296, 2008. doi:10.1137/060664690.
- 7 Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974. doi:10.1016/0012-365X(74)90002-8.
- 8 Derek Gordon Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013. doi:10.1137/11083856X.
- 9 Derek Gordon Corneil, Jérémie Dusart, Michel Habib, and Ekkehard Köhler. On the power of graph searching for cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 30(1):569–591, 2016. doi:10.1137/15M1012396.
- 10 Derek Gordon Corneil and Richard Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008. doi:10.1137/050623498.
- 11 Derek Gordon Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of lexicographic breadth first searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010. doi:10.1016/j.dam.2009.10.001.
- 12 Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.
- 13 Gabriel Andrew Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961. doi:10.1007/BF02992776.
- 14 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In Manfred Nagl, editor, *WG*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 1995. doi:10.1007/3-540-60618-1\_88.
- 15 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. doi:10.1016/0095-8956(74)90094-X.
- 16 Jan Gorzny. *Related Orderings of AT-Free Graphs*. PhD thesis, University of Waterloo, 2022. URL: <https://uwspace.uwaterloo.ca/handle/10012/18079>.

- 17 Torben Hagerup. Biconnected graph assembly and recognition of DFS trees. Technical report, Universität des Saarlandes, 1985. doi:10.22028/D291-26437.
- 18 Torben Hagerup and Manfred Nowak. Recognition of spanning trees defined by graph searches. Technical report, Universität des Saarlandes, 1985.
- 19 Vojtěch Jarník. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti*, 6(4):57–63, 1930. URL: [https://dml.cz/bitstream/handle/10338.dmlcz/500726/Jarnik\\_01-0000-31\\_1.pdf](https://dml.cz/bitstream/handle/10338.dmlcz/500726/Jarnik_01-0000-31_1.pdf).
- 20 Ephraim Korach and Zvi Ostfeld. DFS tree construction: Algorithms and characterizations. In Jan van Leeuwen, editor, *WG*, volume 344 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 1988. doi:10.1007/3-540-50728-0\_37.
- 21 Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001. doi:10.1016/S1571-0653(05)80078-2.
- 22 P. Sreenivasa Kumar and C. E. Veni Madhavan. Minimal vertex separators of chordal graphs. *Discrete Applied Mathematics*, 89(1-3):155–168, 1998. doi:10.1016/S0166-218X(98)00123-1.
- 23 Robert Clay Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. doi:10.1002/j.1538-7305.1957.tb01515.x.
- 24 Guozhen Rong, Yixin Cao, Jianxin Wang, and Zhifeng Wang. Graph searches and their end vertices. *Algorithmica*, 84(9):2642–2666, 2022. doi:10.1007/s00453-022-00981-5.
- 25 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.
- 26 Robert Scheffler. Linearizing partial search orders. In Michael A. Bekos and Michael Kaufmann, editors, *WG*, volume 13453 of *Lecture Notes in Computer Science*, pages 425–438. Springer, 2022. doi:10.1007/978-3-031-15914-5\_31.
- 27 Robert Scheffler. On the recognition of search trees generated by BFS and DFS. *Theoretical Computer Science*, 936:116–128, 2022. doi:10.1016/j.tcs.2022.09.018.
- 28 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.
- 29 Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984. doi:10.1137/0213035.
- 30 James Richard Walte. *Representations of Rigid Cycle Graphs*. PhD thesis, Wayne State University, 1972.
- 31 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, second edition, 2001. URL: [https://openlibrary.org/books/OL6785828M/Introduction\\_to\\_graph\\_theory](https://openlibrary.org/books/OL6785828M/Introduction_to_graph_theory).
- 32 Meibiao Zou, Zhifeng Wang, Jianxin Wang, and Yixin Cao. End vertices of graph searches on bipartite graphs. *Information Processing Letters*, 173:Nr. 106176, 2022. doi:10.1016/j.ipl.2021.106176.



# Probabilistic Input-Driven Pushdown Automata

Alex Rose  

Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

Alexander Okhotin  

Department of Mathematics and Computer Science, Saint Petersburg State University, Russia

---

## Abstract

A probabilistic variant of input-driven pushdown automata (IDPDA), also known as visibly pushdown automata, is introduced. It is proved that these automata can be determinized: an  $n$ -state probabilistic IDPDA that accepts each string with probability at least  $\lambda + \delta$  or at most  $\lambda - \delta$  is transformed to a deterministic IDPDA with at most  $(1 + \frac{1}{\delta})^{n^2 - n}$  states recognizing the same language. An asymptotically close lower bound is provided: for infinitely many  $n$ , there is a probabilistic IDPDA with  $4n + 1$  states and  $\delta = \frac{1}{270n}$ , such that every equivalent deterministic IDPDA needs at least  $7^{n^2/14}$  states. A few special cases of automata with reduced determinization complexity are identified.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Probabilistic computation; Theory of computation  $\rightarrow$  Formal languages and automata theory

**Keywords and phrases** Finite automata, probabilistic automata, input-driven automata, visibly pushdown automata, state complexity

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.78

**Funding** This work was supported by the Russian Science Foundation, project 23-11-00133.

## 1 Introduction

In probabilistic models of computation, there are several options for every step of the computation, each with a specified probability. Probabilistic finite automata (PFA) were introduced by Rabin [23] in 1963. Assuming the two-sided bounded-error condition, that is, that every string is accepted either with probability at most  $\lambda - \delta$ , or with probability at least  $\lambda + \delta$ , for some  $\lambda$  and  $\delta$  with  $0 < \lambda - \delta < \lambda + \delta < 1$ , Rabin proved that every such automaton with  $n$  states can be transformed to a deterministic automaton (DFA) with at most  $(1 + \frac{1}{\delta})^{n-1}$  states. The lower bound on the determinization complexity has been refined several times: in 1982, Freivalds [11] proved that in the worst case  $2^{\Omega(\sqrt{n})}$  states are necessary, in 1996, Ambainis [4] improved this lower bound to  $\Omega(2^{n \frac{\log \log n}{\log n}})$ . Finally, in 2008, Freivalds [12] established the first exponential lower bound of the order  $7^{n/14}$ , improved to  $2^{n/4}$  under Artin's conjecture from number theory.

A probabilistic version of pushdown automata (PDA) was studied by Freivalds [10], who proved that they can recognize a language not recognized by any nondeterministic PDA. Later, Hromkovič and Schnitger [13] showed that these two models are incomparable in power, whereas probabilistic PDA with one-sided error are weaker than both models, yet stronger than deterministic PDA.

The concept of *input-driven pushdown automata* (IDPDA) was introduced by Mehlhorn [16] in 1980. This is a special case of pushdown automata, in which the operations performed on the stack are determined by the input symbols. Nondeterministic IDPDA were first defined by von Braunmühl and Verbeek [25], who proved that they can be determinized, with an  $n$ -state nondeterministic automaton transformed to a deterministic one with  $2^{n^2}$  states and  $O(2^{n^2})$  stack symbols. In 2004, Alur and Madhusudan [2] have reintroduced the model under the name of *visibly pushdown automata*, and obtained some important new



© Alex Rose and Alexander Okhotin;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 78; pp. 78:1–78:14

Leibniz International Proceedings in Informatics



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

results: in particular, they proved that determinization requires  $2^{\Omega(n^2)}$  states in the worst case, established the closure of the family under all basic language operations, and determined the computational complexity of the inclusion and universality problems for IDPDA. Their paper motivated further research in this area. In particular, several related models were investigated, such as alternating [7], unambiguous [21] and timed input-driven pushdown automata [24, 8, 19], input-driven pushdown automata on infinite strings [3, 15, 22]. This research has also inspired some further models, such as pushdown automata driven by finite transducers [14].

In this paper we introduce and study the probabilistic version of IDPDA, with two-sided bounded-error condition, similar to the classical PFA [23] and probabilistic PDA [10]. Even though input-driven pushdown automata with randomized transitions appeared in several papers on automata on infinite strings and their verification [5, 9, 26], the cited papers were not concerned with language recognition, making their models much different from what is studied in this paper.

The first result of this paper is that probabilistic IDPDA with bounded error define the same class of languages as deterministic IDPDA. To be precise, if a language is recognized by bounded-error  $n$ -state PIDPDA with  $\delta$ -cutpoint, then the same language is recognized by a DIDPDA with  $(1 + \frac{1}{\delta})^{n^2-n}$  states and  $\text{const} \cdot (1 + \frac{1}{\delta})^{n^2-n}$  stack symbols. This is done via considering transition matrices on well-nested strings, and proving that if two matrices are close enough with respect to a certain norm, then they are equivalent under an equivalence relation defined by Alur et al. [1]. This in turn implies that the language is recognizable by an IDPDA.

In Section 5 we also give a lower bound. To do so we essentially find a way to lift lower bounds for PFAs to lower bounds for PIDPDAs. Then we apply the best known lower bound for PFAs – the one due to Freivalds [12].

Finally, in Section 6 we study the special case of automata operating on strings nesting depth one. Three cases of automata with reduced determinization complexity are identified. First, there is an analog of *unary* languages, in which every substring enclosed in brackets is unary, for which determinization requires fewer states than in the general case: only  $O((1 + \frac{1}{\delta})^n)$ . We also prove that the same bound holds for automata with  $\Sigma_0 = \{a, b\}$  if the transitions by  $a$  and  $b$  commute. The latter condition means that the automaton “counts” the number of  $a$ s and  $b$ s. Lastly, we show that if the transitions by every symbol in  $\Sigma_0$  are *deterministic*, then the upper bound can also be significantly reduced to  $n^n$ , under no restrictions on the size of  $\Sigma_0$ .

## 2 Deterministic Input-driven Pushdown Automata

A *deterministic input-driven pushdown automaton* (DIDPDA) [3, 16] is a special case of a deterministic pushdown automaton, in which the input alphabet  $\Sigma$  is split into three disjoint sets of *left brackets*  $\Sigma_{+1}$ , *right brackets*  $\Sigma_{-1}$  and *neutral symbols*  $\Sigma_0$ . The type of the input symbol determines the type of the operation with the stack: on a left bracket from  $\Sigma_{+1}$ , then the automaton always pushes one symbol onto the stack; on a right bracket from  $\Sigma_{-1}$ , the automaton must pop one symbol; finally, on a neutral symbol in  $\Sigma_0$ , the automaton may not use the stack. In this paper, symbols from  $\Sigma_{+1}$  and  $\Sigma_{-1}$  are denoted by left and right angled brackets, respectively ( $<$ ,  $>$ ), whereas lower-case Latin letters from the beginning of the alphabet ( $a, b, c, \dots$ ) are used for symbols from  $\Sigma_0$ .

► **Definition 1.** A *deterministic input-driven pushdown automaton* (DIDPDA) is a 6-tuple  $A = (\Sigma, Q, \Gamma, q_0, [\delta_a]_{a \in \Sigma}, F)$ , where

■  $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$  is an input alphabet split into three disjoint classes;

- $Q$  is a finite set of states of the automaton, with an initial state  $q_0 \in Q$  and with a subset of accepting states  $F \subseteq Q$ ;
- $\Gamma$  is a finite set of stack symbols,
- the transition function by each left bracket symbol  $< \in \Sigma_{+1}$  is a function  $\delta_{<}: Q \rightarrow Q \times \Gamma$ , which, for a given current state, provides the next state and the symbol to be pushed onto the stack;
- for every right bracket symbol  $> \in \Sigma_{-1}$ , a function  $\delta_{>}: Q \times \Gamma \rightarrow Q$  specifies the next state, assuming that the given symbol is popped from the stack;
- for a neutral symbol  $c \in \Sigma_0$ , a function  $\delta_c: Q \rightarrow Q$  provides the next state.

For a string  $w = a_1 \dots a_\ell$ , with  $a_1, \dots, a_\ell \in \Sigma$ , the computation on  $w$  starting in a state  $r_0$  with stack contents  $\gamma_0 \in \Gamma^*$  is the sequence  $\{(r_i, \gamma_i)\}_{i=0}^\ell \in Q \times \Gamma^*$ , defined as follows.

- If  $a_i \in \Sigma_{+1}$ , then  $\delta_{a_i}(r_{i-1}) = (r_i, s)$  and  $\gamma_i = \gamma_{i-1}s$ ,
- If  $a_i \in \Sigma_{-1}$ , then  $\gamma_{i-1} = \gamma_i s$ , for some  $s \in \Gamma$ , and  $r_i = \delta_{a_i}(r_{i-1}, s)$ .
- If  $a_i \in \Sigma_0$ , then  $\gamma_i = \gamma_{i-1}$  and  $r_i = \delta_{a_i}(r_{i-1})$ .

If  $w$  is well-nested, then this computation is always defined, and ends in the configuration  $(r_\ell, \gamma_\ell)$ , with  $\gamma_\ell = \gamma_0$ .

A well-nested string  $w$  is accepted if the computation on  $w$  starting in  $q_0$  with the empty stack ends in a configuration  $(q, \varepsilon)$  with  $q \in F$ . The set of all accepted strings is denoted by  $L(A)$ .

One can notice that each word  $w$  defines a function  $f^w: Q \rightarrow Q$ , such that if  $A$  is in state  $q$ , it will be in state  $f^w(q)$  after reading  $w$ . Then  $L(A) = \{w \mid w \text{ is well-nested and } f^w(q_0) \in F\}$ .

For every language  $L$ , we define the following equivalence relation on the set of all well-nested strings.

► **Definition 2.** Let  $L$  be a set of well-nested strings. Let  $w_1$  and  $w_2$  be well-nested. The relation  $\approx_L$  on the set of well-nested strings is defined by  $w_1 \approx_L w_2$  if, for every two strings  $u, v$  with  $uw$  well-nested, the string  $uw_1v$  is in  $L$  if and only if  $uw_2v$  is in  $L$ .

If the language  $L$  is understood from the context, the relation  $\approx_L$  is denoted by  $\approx$ .

► **Lemma 3** (Alur et al. [1]).  $L$  is recognized by an IDPDA if and only if there is only a finite number of the equivalence classes with respect to this equivalence relation.

Moreover, DIDPDA can be chosen such that it has  $m$  states and  $|\Sigma_{+1}| \cdot m$  stack symbols, where  $m$  is the number of the equivalence classes. Also, if  $L$  is recognized by an IDPDA with  $n$  states, then the number of equivalence classes is not more than  $n^n$ .

### 3 Probabilistic Input-driven Automata

Unlike DIDPDAs, probabilistic input-driven pushdown automata may have multiple available transitions, with a probability of making each of them. Nevertheless, whether the automaton pushes, pops or leaves the stack intact, is still determined by the current input symbol. In the end, the input string is accepted if and only if the probability of reaching an accepting state after reading the string is sufficiently large.

► **Definition 4.** Let  $S$  be a countable or a finite set. Let  $D(S)$  denote the set of probability distributions on  $S$ , that is,  $D(S) = \{p: S \rightarrow [0, 1] \mid \sum_{x \in S} p(x) = 1\}$ .

► **Definition 5.** A probabilistic input-driven pushdown automaton (PIDPDA) is a 6-tuple  $A = (\Sigma, Q, \Gamma, q_0, [\delta_a]_{a \in \Sigma}, F)$ , where

## 78:4 Probabilistic Input-Driven Pushdown Automata

- $\Sigma = \Sigma_{+1} \sqcup \Sigma_0 \sqcup \Sigma_{-1}$  is an input alphabet split into three disjoint classes;
- $Q$  is a finite set of states of the automaton, with an initial state  $q_0 \in D(Q)$  and with a subset of accepting states  $F \subseteq Q$ ;
- $\Gamma$  is a finite set of stack symbols,
- the transition function by each left bracket symbol  $< \in \Sigma_{+1}$  is a function  $\delta_{<}: Q \rightarrow D(Q \times \Gamma)$ , which, for a given current state  $p \in Q$ , assigns a probability to each pair  $(q, s)$ , that is, the probability of pushing  $s$  onto the stack and entering the state  $q$ ;
- for every right bracket symbol  $> \in \Sigma_{-1}$ , a function  $\delta_{>}: Q \times \Gamma \rightarrow D(Q)$  specifies the probabilities of entering each state, assuming that the given symbol is popped from the stack;
- for a neutral symbol  $c \in \Sigma_0$ , a function  $\delta_c: Q \rightarrow D(Q)$  provides the probabilities of the next state.

For a string  $w = a_1 \dots a_\ell$ , with  $a_1, \dots, a_\ell \in \Sigma$ , a computation sequence on  $w$  is a sequence  $\{(r_i, \gamma_i)\}_{i=0}^\ell \in Q \times \Gamma^*$ , which satisfies the following conditions.

- if  $a_i \in \Sigma_{+1}$ , then  $\gamma_i = \gamma_{i-1}s$ , for some  $s \in \Gamma$ , and the probability of this step is defined by  $p_i = \delta_{a_i}(r_{i-1})(r_i, s)$ ,
- if  $a_i \in \Sigma_{-1}$ , then  $\gamma_{i-1} = \gamma_i s$ , for some  $s \in \Gamma$ , and the probability of this step is  $p_i = \delta_{a_i}(r_{i-1}, s)(r_i)$ .
- if  $a_i \in \Sigma_0$ , then  $\gamma_i = \gamma_{i-1}$  and the probability is  $\delta_{a_i}(r_{i-1})(r_i)$ .

The probability of such a sequence is the product  $p_1 \cdot \dots \cdot p_\ell$  of probabilities of individual steps.

The probability of going from configuration  $(q, \gamma)$  to configuration  $(q', \gamma')$  by reading a string  $w$  is the sum of probabilities of all computation sequences on  $w$  that start with  $(q, \gamma)$  and end with  $(q', \gamma')$ .

The probability of accepting a well-nested string  $w$  is the probability of going from its initial configuration to any accepting configuration.

$$\Pr(A \text{ accepts } w) = \sum_{q \in F} \Pr(A \text{ goes from } (q_0, \varepsilon) \text{ to } (q, \varepsilon) \text{ by reading } w)$$

► **Definition 6.** A probabilistic input-driven pushdown automaton  $A$  is said to have a  $\delta$ -cut-point  $\lambda$ , with  $\delta > 0$  and  $\lambda \in [0, 1]$ , if, for every well-nested  $w$ , either  $\Pr(A \text{ accepts } w) \geq \lambda + \delta$  or  $\Pr(A \text{ accepts } w) \leq \lambda - \delta$ .

► **Definition 7.** The language  $L(A)$  recognized by an automaton  $A$  with a  $\delta$ -cut-point  $\lambda$  is the set of all well-nested strings  $w$  for which  $\Pr(A \text{ accepts } w) \geq \lambda + \delta$ .

Similarly to the deterministic case, each well-nested word  $w$  defines a stochastic matrix  $P^w$  of order  $|Q| \times |Q|$ , where  $P_{q,r}^w = \Pr(A \text{ goes into } q \text{ from } r \text{ after reading } w)$ . This is a generalization of functions  $f^w$  for deterministic IDPDAs.

### 4 Determinization and Upper Bound

► **Theorem 8.** Let  $A$  be a probabilistic IDPDA with a  $\delta$ -cut-point  $\lambda$  and  $n$  states. Then there exists a deterministic IDPDA that recognizes the same language and has at most  $(1 + \frac{1}{\delta})^{n^2-n}$  states and at most  $|\Sigma_{+1}| \cdot (1 + \frac{1}{\delta})^{n^2-n}$  stack symbols.

To compare, Rabin's [23] transformation of a PFA to a DFA uses only  $(1 + \frac{1}{\delta})^{n-1}$  states. Rabin's argument estimates the number of Myhill–Nerode equivalence classes, which is sufficient to describe the computation of a finite automaton. The computations of input-driven pushdown automata are harder to simulate, and require more involved equivalence classes of Alur et al. ( $\approx$ , see Definition 2).

The main idea of the proof is that if, for two well-nested strings,  $w$  and  $w'$ , the corresponding stochastic matrices  $P^w$  and  $P^{w'}$  are close under a certain metric, then they must be equivalent in the sense of Definition 2, that is, replacing any substring  $w$  with  $w'$  does not change the acceptance status of a string.

We shall use the following lemma which provides an upper bound on the maximum number of stochastic matrices at least  $2\delta$  apart in the metric given by the norm  $\|\xi\| = \max_{1 \leq k \leq n} \sum_{\ell=1}^n |\xi_{k\ell}|$ . The proof uses Rabin's idea involving volumes.

► **Lemma 9.** *Let  $\xi^{(1)}, \dots, \xi^{(m)}$  be stochastic matrices of order  $n$ , such that  $\|\xi^{(\ell)} - \xi^{(k)}\| \geq 2\delta$  for  $\ell \neq k$ . Then  $m \leq (1 + \frac{1}{\delta})^{n^2-n}$ .*

**Proof.** We view  $n \times n$  matrices as points in  $\mathbb{R}^{n^2}$ . Let  $M_n(\mathbb{R}_{\geq 0})$  denote the set of such matrices with non-negative elements. For  $r > 0$  let us define

$$S(r) = \left\{ t \in M_n(\mathbb{R}_{\geq 0}) \mid \sum_{1 \leq j \leq n} t_{ij} = r \text{ for } 1 \leq i \leq n \right\}$$

In particular, it follows from the definition that  $S(1)$  is the set of stochastic matrices.

▷ **Claim 10.** If  $r < \delta$ , then  $\xi^{(k)} + S(r)$  and  $\xi^{(\ell)} + S(r)$  are disjoint for  $k \neq \ell$ .

**Proof.** Suppose  $(\xi^{(k)} + S(r)) \cap (\xi^{(\ell)} + S(r)) \neq \emptyset$ . Then there exist  $x, y \in S(r)$  such that  $\xi^{(k)} + x = \xi^{(\ell)} + y$ . By definition, the norm of every element in  $S(r)$  is equal to  $r$ , therefore  $\|\xi^{(\ell)} - \xi^{(k)}\| = \|x - y\| \leq \|x\| + \|y\| = 2r < 2\delta$ , which contradicts the assumption that  $\|\xi^{(\ell)} - \xi^{(k)}\| \geq 2\delta$ . ◀

Clearly,  $\xi^{(1)}, \dots, \xi^{(m)} \in S(1)$  because they are stochastic. Thus,  $\xi^{(1)} + S(r), \dots, \xi^{(m)} + S(r) \subseteq S(1) + S(r) = S(1+r)$ , where the latter equality follows from the definition of  $S(r)$ . By the claim, the sets  $\xi^{(1)} + S(r), \dots, \xi^{(m)} + S(r)$  are pairwise disjoint, and all of them are contained in  $S(r+1)$ .

Now the plan is to use volumes of the sets  $\xi^{(k)} + S(r)$  to prove that only a limited number of such sets may fit into  $S(1+r)$ . Since the  $n^2$ -dimensional volume of  $S(r)$  is 0, the first step is determine the right dimension. Let  $d$  be the dimension of  $S(r)$ , which is the same as the dimension of  $S(1)$ , since these sets are the same up to scaling. It is claimed that  $d = n^2 - n$ . The set  $S(1)$  is contained in the  $(n^2 - n)$ -dimensional (affine) subspace  $H$  defined by the equations  $t_{i1} + \dots + t_{in} = 1, i = 1, \dots, n$ , so  $d \leq n^2 - n$ . On the other hand,  $S(1)$  contains a  $(n^2 - n)$ -dimensional ball of small radius, confirming that  $d = n^2 - n$ .

Let  $V_d$  denote the  $d$ -dimensional volume. The sum of the volumes of the disjoint sets  $\xi^{(k)} + S(r)$  does not exceed the volume of the set  $S(r+1)$  they are contained in.

$$V_d(S(r+1)) \geq V_d((\xi^{(1)} + S(r)) \cup \dots \cup (\xi^{(m)} + S(r))) = V_d(\xi^{(1)} + S(r)) + \dots + V_d(\xi^{(m)} + S(r))$$

Notice that  $V_d(\xi^{(1)} + S(r)) = V_d(S(r))$  because  $\xi^{(k)} + S(r)$  is a translation of  $S(r)$ . Hence, the last inequality yields  $mV_d(S(r)) \leq V_d(S(r+1))$ .

The linear transformation  $t \mapsto \frac{r+1}{r}t$  maps  $S(r)$  onto  $S(r+1)$  (because if  $\sum_{1 \leq j \leq n} t_{ij} = r$ , then  $\sum_{1 \leq j \leq n} \frac{r+1}{r}t_{ij} = \frac{r+1}{r}r = r+1$ ), therefore  $V_d(S(r+1)) = (\frac{r+1}{r})^d V_d(S(r))$ . But we have already established that  $V_d(S(r+1)) \geq mV_d(S(r))$ , thus  $(1 + \frac{1}{r})^d V_d(S(r)) \geq mV_d(S(r))$  and therefore  $m \leq (1 + \frac{1}{r})^d$  for  $0 < r < \delta$ . Passing to the limit as  $r$  tends to  $\delta$ , we obtain  $m \leq (1 + \frac{1}{\delta})^d = (1 + \frac{1}{\delta})^{n^2-n}$ . ◀

The next lemma provides a connection between the matrices  $P^w$  and the equivalence classes.

► **Lemma 11.** *If  $\|P^{w_1} - P^{w_2}\| < 2\delta$  then  $w_1 \approx w_2$ .*

**Proof.** Consider any well-nested strings  $w$  and  $uv$ . Note that because  $w$  is well-nested, after reading  $uw$  the stack will be exactly the same as after reading  $u$ . Let  $h$  be the nesting depth of  $u$ .

Then  $\Pr(A \text{ accepts } uv)$  is expressed as the following sum over all possible stack contents after reading  $u$  and states before and after reading  $w$ .

$$\sum_{\gamma \in \Gamma^h} \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \Pr(A \text{ goes to } (i, \gamma) \text{ after reading } u) \cdot P_{ij}^w \cdot \Pr(A \text{ accepts from } (j, \gamma) \text{ after reading } v)$$

For brevity, denote the probabilities of parts of this computation by

$$\begin{aligned} q_i(\gamma) &:= \Pr(A \text{ goes into } (i, \gamma) \text{ after reading } u), \\ r_j(\gamma) &:= \Pr(A \text{ accepts from } (j, \gamma) \text{ after reading } v), \end{aligned}$$

so that the above probability is expressed as follows.

$$\Pr(A \text{ accepts } uv) = \sum_{\gamma \in \Gamma^h} \sum_{1 \leq i, j \leq n} q_i(\gamma) \cdot P_{ij}^w \cdot r_j(\gamma)$$

Then the probability  $q_i(\gamma)$  depends only on  $i$ ,  $\gamma$  and  $u$ ; the probability  $r_j(\gamma)$  depends only on  $j$ ,  $\gamma$  and  $v$ ; and  $\sum_{1 \leq i \leq n} \sum_{\gamma \in \Gamma^h} q_i(\gamma) = 1$ .

Therefore, the difference between the probabilities of accepting  $uw_1v$  and  $uw_2v$  is estimated as

$$\begin{aligned} |\Pr(A \text{ accepts } uw_1v) - \Pr(A \text{ accepts } uw_2v)| &= \left| \sum_{\gamma \in \Gamma^h} \sum_{1 \leq i, j \leq n} q_i(\gamma) (P_{ij}^{w_1} - P_{ij}^{w_2}) r_j(\gamma) \right| \leq \\ &\leq \sum_{\gamma \in \Gamma^h} \sum_{1 \leq i, j \leq n} q_i(\gamma) \cdot |P_{ij}^{w_1} - P_{ij}^{w_2}| \cdot r_j(\gamma) \leq \\ &\leq \sum_{\gamma \in \Gamma^h} \sum_{1 \leq i, j \leq n} q_i(\gamma) \cdot |P_{ij}^{w_1} - P_{ij}^{w_2}| = \\ &= \sum_{1 \leq i \leq n} \sum_{\gamma \in \Gamma^h} q_i(\gamma) \sum_{1 \leq j \leq n} |P_{ij}^{w_1} - P_{ij}^{w_2}| \leq \\ &\leq \left( \sum_{1 \leq i \leq n} \sum_{\gamma \in \Gamma^h} q_i(\gamma) \right) \cdot \max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |P_{ij}^{w_1} - P_{ij}^{w_2}| = \\ &= \max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |P_{ij}^{w_1} - P_{ij}^{w_2}| < 2\delta \end{aligned}$$

Since every well-nested string is accepted with the probability of either at least  $\lambda + \delta$  or at most  $\lambda - \delta$ ,  $w_1$  and  $w_2$  are accepted or rejected simultaneously, which proves the claim. ◀

Now we are ready to prove the theorem.

**Proof of Theorem 8.** The proof is by bounding the number of equivalence classes under  $\approx$ . Suppose there are at least  $m$  equivalence classes, then we can take  $w_1, \dots, w_m$  to be the representatives of these classes. This yields  $m$  points  $P^{w_1}, \dots, P^{w_m}$  in  $[0, 1]^{n^2}$ . Due to Lemma 11, for  $k \neq \ell$ , the inequality  $\|P_{ij}^{w_k} - P_{ij}^{w_\ell}\| \geq 2\delta$  holds because  $w_k \not\approx w_\ell$ . Then, Lemma 9 implies  $m \leq (1 + \frac{1}{\delta})^{n^2 - n}$ . By Lemma 3, there is a DIDPDA with  $m$  states and  $|\Sigma_{+1}| \cdot m$  stack symbols accepting the same language. ◀

## 5 Lower bounds

The first exponential lower bounds on the complexity of determinizing an  $n$ -state PFA were constructed by Freivalds [12]. His lower bound exists in two versions, both of the form  $c^n$ , but with different values of  $c$ : one bound holds unconditionally, whereas the other, with a greater base  $c$ , relies on Artin's conjecture from number theory. The second bound also uses PFAs with a smaller error probability.

► **Theorem 12** (Freivalds [12]). *For infinitely many numbers  $n$ , there exists a PFA with  $n$  states with a  $\delta_0$ -cutpoint  $\lambda_0 = \frac{1}{2}$ , such that any equivalent DFA needs at least  $c^n$  states, where  $c = 7^{\frac{1}{14}}$  and  $\delta_0 = \frac{1}{270}$ . If Artin's conjecture is true, then the estimate holds for  $c = 2^{\frac{1}{4}}$ ,  $\delta_0 = \frac{1}{36}$ .*

We will use this result to construct a lower bound in our setting.

► **Theorem 13.** *For infinitely many numbers  $n$ , there exists a PIDPDA with  $4n + 1$  states and a  $\delta_0$ -cutpoint  $\lambda(n) = \frac{1}{2n}$ , such that every equivalent DIDPDA needs at least  $c^{n^2}$  states, where  $c = 7^{\frac{1}{14}}$  and  $\delta_0 = \frac{1}{270}$ . If Artin's conjecture is true, then the estimate holds for  $c = 2^{\frac{1}{4}}$ ,  $\delta_0 = \frac{1}{36}$ .*

**Proof.** For infinitely many numbers  $n$ , Freivalds constructed a language  $K_n$  such that:

- it can be recognized by a PFA  $A$  with a  $\delta_0$ -cutpoint  $\lambda_0 = \frac{1}{2}$  and  $n$  states,
- any DFA recognizing  $K_n$  requires at least  $c^n$  states.

The latter means that there exists a set  $\{u_i \mid 1 \leq i \leq \lceil c^n \rceil\}$  of at least  $c^n$  strings such that for every two strings from this set,  $u_{i_1}$  and  $u_{i_2}$  with  $i_1 \neq i_2$ , there exists a separating string  $v$  with one of the concatenations  $u_{i_1}v$ ,  $u_{i_2}v$  in  $K_n$  and the other not in  $K_n$ . Let  $\{v_j \mid 1 \leq j \leq m\}$  be a finite set of such separating strings, so that for all  $i_1$  and  $i_2$  with  $i_1 \neq i_2$  there exists  $j \in \{1, \dots, m\}$  with  $u_{i_1}v_j \in K_n$  if and only if  $u_{i_2}v_j \notin K_n$ .

In the PFA defined by Freivalds, let  $Q$  be its set of states, and consider the probability distribution on the set of states after reading each string  $u_i$  from the initial state, as well as the probability of accepting each string  $v_j$  from each state, and denote them by the following vectors  $p_i \in \mathbb{R}^n$  and  $r_j \in \mathbb{R}^n$ .

$$(p_i)_q = \Pr(A \text{ goes into } q \text{ after reading } u_i)$$

$$(r_j)_q = \Pr(A \text{ accepts from } q \text{ after reading } v_j)$$

Then the probability of accepting each concatenation  $u_i v_j$  is a scalar product  $\langle p_i, r_j \rangle$ , and since this is a bounded-error PFA with  $\delta_0$ -cutpoint  $\lambda_0 = \frac{1}{2}$ , the following two properties must hold.

- (i) For every  $i, j$  either  $\langle p_i, r_j \rangle \geq \lambda_0 + \delta_0$  or  $\langle p_i, r_j \rangle \leq \lambda_0 - \delta_0$ .
- (ii) For every  $i_1 \neq i_2$  there exists  $j$  such that  $\langle p_{i_1}, r_j \rangle \geq \lambda_0 + \delta_0$  and  $\langle p_{i_2}, r_j \rangle \leq \lambda_0 - \delta_0$  (or  $\langle p_{i_2}, r_j \rangle \geq \lambda_0 + \delta_0$  and  $\langle p_{i_1}, r_j \rangle \leq \lambda_0 - \delta_0$ ).

Now these strings  $u_i$  and  $v_j$ , along with the PFA, are used to construct the desired probabilistic input-driven automaton. It is constructed over an alphabet with a single left bracket,

$$\Sigma_{+1} = \{<\},$$

with a large set of neutral symbols each encoding an  $n$ -tuple of strings  $u_i$ ,

$$\Sigma_0 = \{a_{i_1, \dots, i_n} \mid 1 \leq i_1, \dots, i_n \leq \lceil c^n \rceil\},$$

and with right brackets representing separating strings applied to one particular component of an  $n$ -tuple.

$$\Sigma_{-1} = \{ \rangle_{k,j} \mid k \in \{1, 2, \dots, n\}; 0 \leq j \leq m \}$$

Define the new language as  $L_n = \{ \langle a_{i_1, \dots, i_n} \rangle_{k,j} \mid u_{i_k} v_j \in K_n \}$ . Then, in order to test the membership of a concatenation  $u_{i_k} v_j$  in  $K_n$ , a deterministic IDPDA will have to remember the entire  $n$ -tuple of strings, whereas a probabilistic automaton can randomly choose  $k$  in the beginning, and then simulate Freivalds' automaton on the  $k$ -th component of the tuple.

▷ **Claim 14.** There exists a PIDPDA  $B$  with  $4n + 1$  states and a  $\delta(n)$ -cutpoint  $\lambda(n)$  recognizing  $L_n$ .

*Proof.* An  $n$ -state PIDPDA, which assumes three-symbol input strings of the form  $\langle a_{i_1, \dots, i_n} \rangle_{k,j}$ , is constructed first; later it will be extended to check the form of the string.

The automaton operating on well-formed strings uses the same set of states  $Q$  as Freivalds' automaton. Assume that the states are numbers:  $Q = \{1, \dots, n\}$ , and let  $n$  be the only accepting state. The same set  $Q$  is also used as the stack alphabet.

In the initial state, the automaton reads the left bracket  $\langle$  and equiprobably chooses the next state  $s$  and pushes  $s$  onto the stack. Next, it encounters a symbol  $a_{i_1, \dots, i_n}$  in the state  $s$ , and simply replicates the probability distribution of Freivalds' automaton on the string  $u_{i_s}$ .

$$\delta'_{a_{i_1, \dots, i_n}}(s)(t) = (p_{i_s})_t$$

Finally, upon reading a right bracket  $\rangle_{k,j}$  in a state  $t$ , the automaton should decide whether to enter the accepting state  $n$ . It pops the number  $s$  from the stack, and if it does not match  $k$ , the automaton rejects (by entering  $n$  with probability 0). If  $s$  equals  $k$ , then the automaton accepts with the same probability  $(r_j)_t$ , with which Freivalds' automaton accepts the string  $v_j$  from the state  $t$ .

$$\begin{aligned} \delta'_{\rangle_{k,j}}(t, s)(n) &= \begin{cases} 0, & \text{if } k \neq s \\ (r_j)_t, & \text{if } k = s \end{cases} \\ \delta'_{\rangle_{k,j}}(t, s)(1) &= 1 - \delta'_{\rangle_{k,j}}(t, s)(n) \\ \delta'_{\rangle_{k,j}}(t, s)(\ell) &= 0 \end{aligned} \quad (2 \leq \ell \leq n-1)$$

It is claimed that this automaton accepts a string  $\langle a_{i_1, \dots, i_n} \rangle_{k,j}$  with probability  $\frac{1}{n} \langle p_{i_k}, r_j \rangle$ . Indeed, the randomly chosen number  $s$  matches  $k$  with probability  $\frac{1}{n}$ , and, provided that it happened, the probability of acceptance is

$$\begin{aligned} &\sum_{q \in Q} \left( \Pr(B \text{ goes from } k \text{ to } q \text{ after reading } a_{i_1, \dots, i_n}) \cdot \right. \\ &\quad \left. \Pr(B \text{ accepts from } q \text{ upon reading } \rangle_{k,j} \text{ with } k \text{ in the stack}) \right) = \\ &= \sum_{q \in Q} (p_{i_k})_q \cdot (r_j)_q = \langle p_{i_k}, q_j \rangle \end{aligned}$$

By (ii), the scalar product  $\langle p_{i_k}, q_j \rangle$  is either at least  $\lambda_0 + \delta_0$  (if  $u_{i_k} v_j \in K_n$ ), or at most  $\lambda_0 - \delta_0$  (if  $u_{i_k} v_j \notin K_n$ ). Therefore, the overall probability is either at least  $\frac{\lambda_0}{n} + \frac{\delta_0}{n}$ , or at most  $\frac{\lambda_0}{n} - \frac{\delta_0}{n}$ .

The construction above works for words in  $\langle \Sigma_0 \Sigma_{-1}$ , other words may violate the  $\delta$ -cutpoint condition. To eliminate them we can take a 4-state partial DFA that verifies that the input string is indeed of the form  $\langle \Sigma_0 \Sigma_{-1}$ , and take the direct product of the automaton above and this DFA. One extra dead state is added for ill-formed strings. ◁

▷ **Claim 15.** Any DIDPDA recognizing  $L_n$  has at least  $|\Sigma_0| \geq c^{n^2}$  states.

*Proof.* Since all well-formed strings begin with the same left bracket  $\langle$ , a deterministic automaton cannot store any information on the stack: it always pushes the same stack symbol. Consider the state of the DIDPDA after reading  $\langle a_{i_1, \dots, i_n}$ . It is claimed that this state must be different for different symbols.

Suppose the contrary, that for some two symbols  $a_{i_1, \dots, i_n} \neq a_{i'_1, \dots, i'_n}$ , the state after reading them is the same state  $q$ . Let  $k$  be such that  $i_k \neq i'_k$ , and let  $v_j$  be the separating string for  $u_{i_k}$  and  $u_{i'_k}$  with respect to Freivalds' automaton, that is, exactly one of the strings  $u_{i_k} v_j$ ,  $u_{i'_k} v_j$  is in  $K_n$ . Then, by the definition of  $L_n$ , exactly one of the strings  $\langle a_{i_1, \dots, i_n} \rangle_{k,j}$  and  $\langle a_{i'_1, \dots, i'_n} \rangle_{k,j}$  is in  $L_n$ . However, since the IDPDA is in the same configuration before reading  $\rangle_{k,j}$  on either string, it either accepts both strings or rejects both of them. The contradiction obtained shows that the number of states in the DIDPDA is at least  $|\Sigma_0|$ . ◀

The theorem follows from the two claims. ◀

The upper bound on the size of the constructed automaton gives  $(1 + \frac{1}{\delta(n)})^{(4n+1)^2 - (4n+1)} = (1 + \frac{n}{\delta_0})^{16n^2 + 4n} \leq (\frac{n+1}{\delta_0})^{16n^2 + 4n} = c^{(16n^2 + 4n) \log_c \frac{n+1}{\delta_0}} = c^{(16n^2 + 4n)(\log_c(n+1) - \log_c \delta_0)} = c^{(16+o(1))n^2 \log_c n} = c^{(16 \log_c 2 + o(1))n^2 \log_2 n}$ . We see that the exponent differs from the one in the lower bound by an  $O(\log n)$  factor.

## 6 Sharper Upper Bounds in Special Cases

For probabilistic finite automata, the case of a unary alphabet is much different from the general case. The first lower bound on the determinization complexity in the unary case was given by Freivalds [11]. Milani and Pighizzini [18] proved that the worst-case determinization blowup in the unary case is of the order of Landau's function, that is,  $e^{(1+o(1))\sqrt{n} \ln n}$ . Later Mereghetti et al. [17] and Bianchi et al. [6] investigated more details of the complexity of unary PFAs.

For input-driven pushdown automata, there is no unary case in the strict sense: as long as there is a pair of matching brackets, one can use them to encode any alphabet. In order to obtain a variant of the unary case, the use of brackets should be somehow restricted. The following condition of *nesting depth one* still allows encoding a binary alphabet by abusing the brackets, but this can be done only outside the brackets; if  $|\Sigma_0| = 1$ , then inside the brackets the string is truly unary.

► **Definition 16.** A well-nested language  $L$  is called a *depth-one language* if for every  $w \in L$  the maximal nesting depth of  $w$  is one, i.e.,  $L \subseteq \Sigma_0^*(\Sigma_{+1}\Sigma_0^*\Sigma_{-1}\Sigma_0^*)^*$ .

For depth-one languages, it is natural to consider the classical Myhill–Nerode relation operating on the outer level of brackets and restricted to well-nested strings.

► **Definition 17.** For a language of well-nested strings  $L$ , define a relation  $\sim_L$  on the set of well-nested strings by  $u \sim_L u'$  if and only if, for every well-nested string  $v$ , the string  $uv$  is in  $L$  if and only if  $u'v$  is in  $L$ . When the language  $L$  is clear from the context, the relation  $\sim_L$  shall be denoted by  $\sim$ .

Denote by  $[u]$  the equivalence class of a string  $u$  under  $\sim$ .

Later it will be shown that if a depth-one language uses unary strings inside the brackets, then the determinization complexity is reduced. More generally, assume that the string inside each pair of brackets belongs to a regular set  $S \subseteq \Sigma_0^*$ .

► **Definition 18.** Let  $S \subseteq \Sigma_0^*$  be a regular prefix-closed language, that is, if  $w_1w_2 \in S$ , then  $w_1 \in S$ . A depth-one language  $L$  is called  $S$ -nice if, for every string  $x \in L$  and for every partition  $x = u\langle w \rangle v$ , with  $w \in \Sigma_0^*$ , the string  $w$  is in  $S$ .

For an  $S$ -nice language  $L$ , consider the equivalence relation from Definition 2, defined by  $w \approx w'$  if and only if, for all  $x, y$  with  $xy$  well-nested,  $xwy \in L$  if and only if  $xw'y \in L$ . The following notation is introduced for equivalence classes restricted to elements of  $S$ .

► **Definition 19.** For  $u \in S$ , let  $[[u]]_S \subseteq S$  be the set of all strings in  $S$  equivalent to  $u$  under the relation  $\approx$ .

For  $S$ -nice languages, there is an automaton of size proportional to the number of these equivalence classes (cf. Lemma 3 for the general case).

► **Lemma 20.** Let  $L$  be an  $S$ -nice language, and let  $n_1$  be the number of equivalence classes under  $\sim$ , and let  $n_2$  be the number of equivalence classes under  $\approx$  restricted to  $S$ . Then there exists a DIDPDA recognizing  $L$  with  $n_1 + O(n_2)$  states and  $n_1 \cdot |\Sigma_{+1}|$  stack symbols.

Equivalence classes under  $\sim$  become states used outside brackets. Whenever a bracket is encountered, these states are pushed onto the stack along with the bracket. Equivalence classes under  $\approx$  are used as states inside brackets. The details are omitted for brevity.

Thus, in order to obtain upper bounds on the size of DIDPDA recognizing various languages of restricted form, it is sufficient to estimate the number of equivalence classes under both relations  $\sim$  and  $\approx$ .

The first observation is that the number of Myhill–Nerode classes on the outer level of brackets can be estimated using Rabin’s [23] argument for finite automata.

► **Lemma 21.** Let a language  $L$  be recognized by a PIDPDA  $A$  with  $n$  states and a  $\delta$ -cutpoint  $\lambda$ . Then  $L$  has at most  $(1 + \frac{1}{\delta})^{n-1}$  equivalence classes under  $\sim$ .

**Sketch of a proof.** Rabin’s [23, Thm. 3] argument works, because it never refers to actual transitions of a probabilistic automaton, and uses only probabilities of computations over prefixes and suffixes. If these prefixes and suffixes contain any brackets, this does not affect the argument.

For well-nested  $u$  and  $v$  we introduce the vectors  $p(u), r(v) \in \mathbb{R}^n$  with

$$p(u)_i = \Pr(A \text{ goes into } q_i \text{ after reading } u),$$

$$r(v)_i = \Pr(A \text{ accepts from } q_i \text{ after reading } v).$$

Then the probability that  $A$  accepts  $uv$  is equal to

$$\sum_{i=1}^n \Pr(A \text{ goes into } q_i \text{ after reading } u) \cdot \Pr(A \text{ accepts from } q_i \text{ after reading } v) = \langle p(u), r(v) \rangle.$$

Again, for well-nested  $u$  and  $u'$  it turns out that  $\sum_{i=1}^n |p(u)_i - p(u')_i| < 2\delta$  implies  $u \sim u'$ , because for any well-nested  $v$  the following inequality holds.

$$\begin{aligned} |\Pr(A \text{ accepts } uv) - \Pr(A \text{ accepts } u'v)| &= |\langle p(u), r(v) \rangle - \langle p(u'), r(v) \rangle| = \\ &= |\langle p(u) - p(u'), r(v) \rangle| = \left| \sum_{i=1}^n (p(u)_i - p(u')_i) r(v)_i \right| \leq \\ &\leq \sum_{i=1}^n |p(u)_i - p(u')_i| \cdot r(v)_i \leq \sum_{i=1}^n |p(u)_i - p(u')_i| < 2\delta \end{aligned}$$

Therefore, pairwise inequivalent strings  $u_1, \dots, u_m$  yield vectors  $p(u_1), \dots, p(u_m)$  with  $\sum_{i=1}^n |p(u_k)_i - p(u_l)_i| > 2\delta$ , and, as Rabin showed, in this case  $m \leq (1 + \frac{1}{\delta})^n$ . ◀

In some special cases of languages, the number of equivalence classes under  $\approx$  restricted to  $S$  is fairly small, leading to upper bounds on the size of DIDPDA. The first such case is when the probabilistic automaton behaves deterministically inside the brackets.

► **Theorem 22.** *Let  $A$  be a PIDPDA with  $n$  states and a  $\delta$ -cutpoint  $\lambda$  that recognizes a depth-one language, and assume that all transitions by symbols in  $\Sigma_0$  are deterministic. Then there is an equivalent DIDPDA with  $O(n^n)$  states.*

**Proof.** By Lemma 21, the number of equivalence classes under  $\sim$  for  $L$  is at most  $(1 + \frac{1}{\delta})^{n-1}$ .

The language is  $S$ -nice with  $S = \Sigma_0^*$ . Recall that if  $P^{w_1} = P^{w_2}$ , the words  $w_1$  and  $w_2$  are equivalent under  $\approx$ . Since all matrices  $P^w$  for  $w \in S = \Sigma_0^*$  are deterministic, they correspond to functions from  $Q$  to  $Q$  and, hence, the number of such matrices is not greater than  $n^n$ . Thus, there are at most  $n^n$  equivalence classes under  $\approx$  restricted to  $S$ .

Finally, by Lemma 20, there is a DIDPDA with  $O((1 + \frac{1}{\delta})^{n-1} + n^n) = O(n^n)$  states. ◀

The theorem implies that it is impossible to achieve the  $\Omega(c^{n^2})$  lower bound with a depth-one language using probabilistic transitions only for the brackets.

The special case of automata in Theorem 22 had the transition matrices inside the brackets generate a finite set of size  $n^n$ . In other special cases of automata with reduced determinization complexity, defined below, transition matrices inside the brackets generate infinite subspaces, yet the *dimension* of those subspaces is bounded. The following lemma allows a small DIDPDA to be constructed in such cases.

► **Lemma 23.** *Let  $L$  be an  $S$ -nice language recognized by a PIDPDA with  $n$  states. Let  $W$  be the subspace of  $M_n(\mathbb{R})$  generated by  $\{P^w \mid w \in S\}$ . Then there is a DIDPDA with  $O((1 + \frac{1}{\delta})^{\max\{n-1, \dim W\}})$  states recognizing  $L$ .*

The proof of Lemma 23 relies on the following geometric property.

► **Lemma 24.** *Let  $\xi^{(1)}, \dots, \xi^{(m)}$  be stochastic matrices of order  $n$ , such that  $\|\xi^{(k)} - \xi^{(\ell)}\| \geq 2\delta$  for  $k \neq \ell$ . Assume that there exists a linear subspace  $W \leq M_n(\mathbb{R})$ , such that for every  $i$  the matrix  $\xi^{(i)}$  lies in  $W$ . Then  $m \leq (1 + \frac{1}{\delta})^{\min\{n^2 - n, \dim W\}}$ .*

Lemma 24 is proved generally similarly to Lemma 9, but requires a more careful choice of  $S(r)$ ; the proof is omitted due to space constraints.

**Proof of Lemma 23.** Indeed, the number of equivalence classes under  $\approx$  is bounded by  $(1 + \frac{1}{\delta})^{\max\{n-1, \dim W\}}$ : if we have  $m$  equivalence classes under  $\approx$ , then they yield  $m$  pairwise inequivalent strings  $w_1, \dots, w_m$ . That, in turn, by Lemma 11, gives rise to  $m$  matrices  $P^{w_1}, \dots, P^{w_m}$  with  $\|P^{w_k} - P^{w_\ell}\| \geq 2\delta$  for  $k \neq \ell$ . By Lemma 24,  $m \leq (1 + \frac{1}{\delta})^{\max\{n-1, \dim W\}}$ . Furthermore, the number of equivalence classes under  $\sim$  is bounded by  $(1 + \frac{1}{\delta})^{n-1}$  by Lemma 21. Combining these two observations and using Lemma 20, we obtain the desired result. ◀

The first class of languages with an improved bound on the dimension of the subspace generated by transition matrices inside the brackets is the following variant of unary languages.

► **Theorem 25.** *If  $L$  is an  $S$ -nice language, and  $S = a_1^* \cup \dots \cup a_k^*$  for some  $a_1, \dots, a_k \in \Sigma_0^*$ , then there is a DIDPDA recognizing  $L$  with  $O((1 + \frac{1}{\delta})^{kn})$  states.*

**Proof.** For every word  $w$  in  $S$  the matrix  $P^w$  is of the form  $(P^{a_i})^m$  for some  $m \in \mathbb{N}_0$  and  $1 \leq i \leq k$ . By Cayley–Hamilton theorem, for each  $a_i$  the space  $W_i$  generated by  $\{(P^{a_i})^m \mid m \geq 0\}$  has the dimension of at most  $n$ ; therefore, all such matrices lie in a vector space  $W = W_1 + \dots + W_n$  such that  $\dim W \leq \sum_{i=1}^n \dim W_i \leq kn$ . It remains to apply Lemma 23. ◀

► **Corollary 26.** *If  $S = a^*$ , then there is a DIDPDA recognizing  $L$  with  $O((1 + \frac{1}{\delta})^n)$  states.*

In particular, the theorem shows that we would not be able to prove the lower bound from Section 5 using a “unary” depth-one language. For instance, the determinization of the automaton that reads  $\langle a^n \rangle$  and verifies that  $n$  belongs to some fixed subset yields at most exponential growth in the number of states. This is somewhat similar to the case of probabilistic finite automata, where the state complexity of the determinization in the unary case is also reduced.

The last special case with improved determinization is the case of automata that use strings over an alphabet  $\{a, b\}$  inside the brackets, and the transitions by  $a$  commute with transitions by  $b$ , that is,  $P^a P^b = P^b P^a$ . In other words, such automata only count the number of  $a$ s and  $b$ s inside the brackets. In this case, there is the following known result on the dimension of the subspace they generate.

► **Theorem 27** (Gerstenhaber, 1961). *Let  $A, B \in M_n(\mathbb{R})$  be a pair of commuting matrices. Then the dimension of the subalgebra generated by  $\{A, B\}$  is at most  $n$ .*

► **Theorem 28.** *If  $S = \{a, b\}^*$  and  $P^a$  and  $P^b$  commute, then there exists an equivalent DIDPDA with  $O((1 + \frac{1}{\delta})^n)$  states.*

**Proof.** If  $w \in S$ , then  $P^w = (P^a)^{|w|_a} (P^b)^{|w|_b}$  lies in the subalgebra generated by  $\{P^a, P^b\}$ , whose dimension is at most  $n$ . Now we can use Lemma 23 to get the desired upper bound. ◀

## 7 Conclusion

It would be interesting to refine the results on the complexity of determinization for the new model by proving a lower bound on both the number of states and the number of stack symbols. Such a lower bound is known for the determinization of nondeterministic input-driven pushdown automata [20] and of their event-clock real-time extension [19]. The method employed in these papers uses strings of arbitrarily large nesting depth, and the automaton makes non-deterministic choices at each nesting level; however, if the same approach were used in our case, then the probability of error would tend to 1 as the nesting depth goes to infinity. Apparently, a new method would be necessary to prove such a bound in the probabilistic case.

Another interesting direction to pursue is improving the bound with respect to  $\delta$ . Our current upper bound is polynomial in  $\frac{1}{\delta}$ . However, it seems possible that there is room for improvement: either a trade-off between the number of states and the probability of error, or perhaps an upper bound that does not depend on  $\delta$  at all.

---

## References

- 1 Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005. doi:10.1007/11523468\_89.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.
- 3 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

- 4 Andris Ambainis. The complexity of probabilistic versus deterministic finite automata. In Tetsuo Asano, Yoshihide Igarashi, Hiroshi Nagamochi, Satoru Miyano, and Subhash Suri, editors, *Algorithms and Computation, 7th International Symposium, ISAAC '96, Osaka, Japan, December 16-18, 1996, Proceedings*, volume 1178 of *Lecture Notes in Computer Science*, pages 233–238. Springer, 1996. doi:10.1007/BFb0009499.
- 5 Nathalie Bertrand, Serge Haddad, and Engel Lefaucheu. Diagnosis in infinite-state probabilistic systems. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPICs*, pages 37:1–37:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CONCUR.2016.37.
- 6 Maria Paola Bianchi, Carlo Mereghetti, Beatrice Palano, and Giovanni Pighizzini. On the size of unary probabilistic and nondeterministic automata. *Fundam. Informaticae*, 112(2-3):119–135, 2011. doi:10.3233/FI-2011-583.
- 7 Laura Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 – Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2007. doi:10.1007/978-3-540-74407-8\_32.
- 8 Laura Bozzelli, Aniello Murano, and Adriano Peron. Event-clock nested automata. In Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira, editors, *Language and Automata Theory and Applications – 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings*, volume 10792 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2018. doi:10.1007/978-3-319-77313-1\_6.
- 9 Vojtech Forejt, Petr Jancar, Stefan Kiefer, and James Worrell. Game characterization of probabilistic bisimilarity, and applications to pushdown automata. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:13)2018.
- 10 Rusins Freivalds. Language recognition using probabilistic turing machines in real time, and automata with a push-down store. *Probl. Inform. Transm.*, 15(4):319–323, 1979.
- 11 Rusins Freivalds. On the growth of the number of states in result of determinization of probabilistic finite automata. *Avtomatika i Viciskitelnaja Tehnika*, 3:39–42, 1982.
- 12 Rusins Freivalds. Non-constructive methods for finite probabilistic automata. *Int. J. Found. Comput. Sci.*, 19(3):565–580, 2008. doi:10.1142/S0129054108005826.
- 13 Juraj Hromkovic and Georg Schnitger. On probabilistic pushdown automata. *Inf. Comput.*, 208(8):982–995, 2010. doi:10.1016/j.ic.2009.11.001.
- 14 Martin Kutrib, Andreas Malcher, and Matthias Wendlandt. Tinput-driven pushdown, counter, and stack automata. *Fundam. Informaticae*, 155(1-2):59–88, 2017. doi:10.3233/FI-2017-1576.
- 15 Christof Löding, P. Madhusudan, and Olivier Serre. Visibly pushdown games. 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 408–420. Springer, 2004. doi:10.1007/978-3-540-30538-5\_34.
- 16 Kurt Mehlhorn. Pebbling mountain ranges and its application of dcfl-recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordwijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2\_89.
- 17 Carlo Mereghetti, Beatrice Palano, and Giovanni Pighizzini. Note on the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata. *RAIRO Theor. Informatics Appl.*, 35(5):477–490, 2001. doi:10.1051/ita:2001106.

- 18 Massimiliano Milani and Giovanni Pighizzini. Tight bounds on the simulation of unary probabilistic automata by deterministic automata. *J. Autom. Lang. Comb.*, 6(4):481–492, 2001. doi:10.25596/jalc-2001-481.
- 19 Mizuhito Ogawa and Alexander Okhotin. On the determinization of event-clock input-driven pushdown automata. In Alexander S. Kulikov and Sofya Raskhodnikova, editors, *Computer Science – Theory and Applications – 17th International Computer Science Symposium in Russia, CSR 2022, Virtual Event, June 29 – July 1, 2022, Proceedings*, volume 13296 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 2022. doi:10.1007/978-3-031-09574-0\_16.
- 20 Alexander Okhotin, Xiaoxue Piao, and Kai Salomaa. Descriptive complexity of input-driven pushdown automata. In Henning Bordihn, Martin Kutrib, and Bianca Truthe, editors, *Languages Alive – Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday*, volume 7300 of *Lecture Notes in Computer Science*, pages 186–206. Springer, 2012. doi:10.1007/978-3-642-31644-9\_13.
- 21 Alexander Okhotin and Kai Salomaa. Descriptive complexity of unambiguous input-driven pushdown automata. *Theor. Comput. Sci.*, 566:1–11, 2015. doi:10.1016/j.tcs.2014.11.015.
- 22 Alexander Okhotin and Victor L. Selivanov. Input-driven pushdown automata on well-nested infinite strings. In Rahul Santhanam and Daniil Musatov, editors, *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 – July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2021. doi:10.1007/978-3-030-79416-3\_21.
- 23 Michael O. Rabin. Probabilistic automata. *Inf. Control.*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.
- 24 Nguyen Van Tang and Mizuhito Ogawa. Event-clock visibly pushdown automata. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24–30, 2009. Proceedings*, volume 5404 of *Lecture Notes in Computer Science*, pages 558–569. Springer, 2009. doi:10.1007/978-3-540-95891-8\_50.
- 25 Burchard von Braunmühl and Rutger Verbeek. Input driven languages are recognized in log n space. In Marek Karplinski and Jan van Leeuwen, editors, *Topics in the Theory of Computation*, volume 102 of *North-Holland Mathematics Studies*, pages 1–19. North-Holland, 1985. doi:10.1016/S0304-0208(08)73072-X.
- 26 Tobias Winkler, Christina Gehnen, and Joost-Pieter Katoen. Model checking temporal properties of recursive probabilistic programs. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures – 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 449–469. Springer, 2022. doi:10.1007/978-3-030-99253-8\_23.

# Counting Homomorphisms from Hypergraphs of Bounded Generalised Hypertree Width: A Logical Characterisation

Benjamin Scheidt ✉ 

Humboldt-Universität zu Berlin, Germany

Nicole Schweikardt ✉ 

Humboldt-Universität zu Berlin, Germany

---

## Abstract

We introduce the 2-sorted counting logic  $\text{GC}^k$  and its restriction  $\text{RGC}^k$  that express properties of hypergraphs. These logics have available  $k$  variables to address hyperedges, an unbounded number of variables to address vertices of a hypergraph, and atomic formulas  $E(e, v)$  to express that a vertex  $v$  is contained in a hyperedge  $e$ . We show that two hypergraphs  $H, H'$  satisfy the same sentences of the logic  $\text{RGC}^k$  if, and only if, they are homomorphism indistinguishable over the class of hypergraphs of generalised hypertree width at most  $k$ . Here,  $H, H'$  are called homomorphism indistinguishable over a class  $\mathcal{C}$  if for every hypergraph  $G \in \mathcal{C}$  the number of homomorphisms from  $G$  to  $H$  equals the number of homomorphisms from  $G$  to  $H'$ . This result can be viewed as a lifting (from graphs to hypergraphs) of a result by Dvořák (2010) stating that any two (undirected, simple, finite) graphs  $H, H'$  are indistinguishable by the  $k+1$ -variable counting logic  $C^{k+1}$  if, and only if, they are homomorphism indistinguishable over the class of graphs of tree-width at most  $k$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Finite Model Theory; Mathematics of computing  $\rightarrow$  Hypergraphs

**Keywords and phrases** counting logics, guarded logics, homomorphism counting, hypertree decompositions, hypergraphs, incidence graphs, quantum graphs

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.79

**Related Version** *Extended Version*: <https://arxiv.org/abs/2303.10980> [22]

**Funding** *Nicole Schweikardt*: Partially supported by the ANR project EQUUS ANR-19-CE48-0019; funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).

**Acknowledgements** We thank Isolde Adler for pointing us to the results in [2, 1, 3], which led to Theorem 2.3.

## 1 Introduction

Counting homomorphisms from a given class  $\mathcal{C}$  of graphs induces a similarity measure between graphs: Consider an arbitrary graph  $H$ . The results of the homomorphism counts for all  $G \in \mathcal{C}$  in  $H$  can be represented by a mapping (or, “vector”)  $\text{HOM}_{\mathcal{C}}(H)$  that associates with every  $G \in \mathcal{C}$  the number  $\text{hom}(G, H)$  of homomorphisms from  $G$  to  $H$ . A similarity measure for the mappings  $\text{HOM}_{\mathcal{C}}(H)$  and  $\text{HOM}_{\mathcal{C}}(H')$  can then be viewed as a similarity measure of two given graphs  $H, H'$ . An overview of this approach, its relations to *graph neural networks*, and its usability as a similarity measure of graphs can be found in Grohe’s survey [17].

Two graphs  $H, H'$  are viewed as “equivalent” (or, *indistinguishable*) over  $\mathcal{C}$  if  $\text{HOM}_{\mathcal{C}}(H) = \text{HOM}_{\mathcal{C}}(H')$ , i.e., for every graph  $G$  in  $\mathcal{C}$  the number of homomorphisms from  $G$  to  $H$  equals the number of homomorphisms from  $G$  to  $H'$ . A classical result by Lovász [19] shows that



© Benjamin Scheidt and Nicole Schweikardt;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 79; pp. 79:1–79:15

Leibniz International Proceedings in Informatics



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

two graphs  $H, H'$  are indistinguishable over the class of *all* graphs if, and only if, they are isomorphic. This inspired a lot of research in recent years, examining the notion of *homomorphism indistinguishability over a class  $\mathcal{C}$*  for various classes  $\mathcal{C}$  [11, 10, 6, 5, 7, 16, 21]. In particular, Grohe [16] proved that two graphs are homomorphism indistinguishable over the class of graphs of *tree-depth*  $\leq k$  if, and only if, they are indistinguishable by sentences of first-order counting logic  $C$  of quantifier-rank  $\leq k$  ( $C$  is the extension of first-order logic with counting quantifiers of the form  $\exists^{\geq n}x$  meaning “there exist at least  $n$  elements  $x$ ”). A decade earlier, Dvořák [11] proved that two graphs are homomorphism indistinguishable over the class of graphs of *tree-width*  $\leq k$  if, and only if, they are indistinguishable by sentences of the  $k+1$ -variable fragment  $C^{k+1}$  of  $C$ . From Cai, Fürer, Immerman [8] we know that this precisely coincides with indistinguishability by the  $k$ -dimensional Weisfeiler-Leman algorithm.

An obvious question is if and how these results can be lifted from graphs to hypergraphs. A first answer was given by Böker in [7]: He introduces a new version of a *color refinement* algorithm on hypergraphs and proves that two hypergraphs  $H, H'$  cannot be distinguished by this algorithm if, and only if, they are homomorphism indistinguishable over the class of *Berge-acyclic* hypergraphs. This is a lifting – from graphs to hypergraphs – of the result of [11, 8] for the case  $k = 1$  (i.e., trees) to “tree-like” hypergraphs. Note that there are different concepts of “tree-likeness” for hypergraphs. Berge-acyclicity is a rather restricted one; it is subsumed by the more general concept of  $\alpha$ -acyclic hypergraphs, which coincides with the hypergraphs of *generalised hypertree width 1* (cf., [13, 14, 12]).

This paper gives a further answer to the above question: For arbitrary  $k \geq 1$  let  $GHW_k$  be the class of hypergraphs of generalised hypertree width  $\leq k$ . Our main result provides a logical characterisation of homomorphism indistinguishability over the class  $GHW_k$ . We introduce a new logic called  $GC^k$  and a restriction  $RGC^k$  of  $GC^k$  and show that two hypergraphs are homomorphism indistinguishable over  $GHW_k$  if, and only if, they are indistinguishable by sentences of the logic  $RGC^k$ .

$GC^k$  is a 2-sorted counting logic for expressing properties of hypergraphs. It has available  $k$  “blue” variables to address edges, and an unbounded number of “red” variables to address vertices of a hypergraph, and atomic formulas  $E(e, v)$  to express that vertex  $v$  is contained in edge  $e$ , as well as atomic formulas  $e = e'$  and  $v = v'$  for expressing equality of edge or vertex variables. Counting quantifiers are of the form  $\exists^{\geq n}\bar{z}$  where  $\bar{z} = (z_1, \dots, z_\ell)$  is either a tuple of edge variables or a tuple of vertex variables; and their meaning is “there exist at least  $n$  tuples  $\bar{z}$ ”. In the logic  $GC^k$ , each vertex variable  $v$  has to be *guarded* by an edge variable  $e$  and an atomic statement  $E(e, v)$  (meaning that vertex  $v$  is included in edge  $e$ ); the use of quantifiers is restricted in a way to ensure that guards are always present. Our design of the logic  $GC^k$  is somewhat inspired by the *guarded fragment of first-order logic* (cf., [4, 15, 14]).  $RGC^k$  imposes certain restrictions on the way guards of red variables can change between quantifications in a formula.

Our main result can be viewed as a lifting – from graphs to hypergraphs – of Dvořák’s [11] result: Dvořák proves that two graphs are homomorphism indistinguishable over the class  $TW_k$  of graphs of *tree-width*  $\leq k$  iff they are indistinguishable by the logic  $C^{k+1}$ . We prove that two hypergraphs are homomorphism indistinguishable over the class  $GHW_k$  of hypergraphs of generalised hypertree width  $\leq k$  iff they are indistinguishable by the logic  $RGC^k$ . This is analogous (although not tightly related) to the following classical results: Kolaitis and Vardi [18] proved that the conjunctive queries of *tree-width*  $\leq k$  are precisely the queries expressible in the  $k+1$ -variable fragment of a certain subclass  $L$  of first-order logic. Gottlob et al. [14] proved that the conjunctive queries of *hypertree width*  $\leq k$  are precisely the ones expressible in the  $k$ -guarded fragment of  $L$ . This is somehow parallel to our result lifting Dvořák’s characterisation; it is what initially gave us the confidence to work on our hypothesis.

The proof of our theorem is at its core very similar to Dvořák’s proof – but it is far from straightforward. Before being able to follow along the lines of Dvořák’s proof, we first have to perform a number of reduction steps and build the necessary machinery. The first step is to move over from homomorphisms on hypergraphs to homomorphisms on incidence graphs. Fortunately, Böker [7] already implicitly achieved what is needed in our setting. The result is: Two hypergraphs  $H, H'$  are homomorphism indistinguishable over the class  $GHW_k$  iff their incidence graphs  $I, I'$  are homomorphism indistinguishable over the class  $IGHW_k$  of incidence graphs of generalised hypertree width  $\leq k$ ; see Section 3.

Next, for an inductive proof in the spirit of Dvořák, we would need an inductive characterisation of the class  $IGHW_k$  in the spirit of [9]. Unfortunately, generalised hypertree decompositions seem to be unsuitable for such a characterisation. That is why we work with severely restricted decompositions that we call *entangled hypertree decompositions* (ehds). In Section 4 we prove that homomorphism indistinguishability over the class  $IGHW_k$  coincides with homomorphism indistinguishability over the class  $IEHW_k$  of incidence graphs of *entangled hypertree width*  $\leq k$ . In our opinion this is interesting on its own, since the requirements of ehds are quite harsh and  $IEHW_k \subsetneq IGHW_k$  for arbitrarily large  $k$ .

In Section 6 we introduce the logic  $\text{GC}^k$  and its restriction  $\text{RGC}^k$ . The inductive characterisation of  $IEHW_k$  follows in Section 7, where we also provide the machinery of *quantum* incidence graphs as an analogue of the quantum graphs used in Dvořák’s proof, tailored towards our setting. In Section 8 we prove that two incidence graphs  $I, I'$  are indistinguishable by the logic  $\text{RGC}^k$  if, and only if, they are homomorphism indistinguishable over the class  $IEHW_k$ . This is achieved by two inductive proofs: We use the inductive characterisation of  $IEHW_k$  to show that for every incidence graph  $J$  in  $IEHW_k$  and every  $m \in \mathbb{N}$  there exists an  $\text{RGC}^k$ -sentence that is satisfied by an incidence graph  $I$  iff there are precisely  $m$  homomorphisms from  $J$  to  $I$ . For the opposite direction, we proceed by induction on the definition of  $\text{RGC}^k$  and construct for every sentence  $\chi$  in  $\text{RGC}^k$  and certain size parameters  $m, d \in \mathbb{N}$  a quantum incidence graph  $Q$  in  $IEHW_k$  satisfying the following: for all incidence graphs  $I$  that match the size parameters  $m, d$ , the number  $\text{hom}(Q, I)$  of homomorphisms from  $Q$  to  $I$  is either 0 or 1, and it is 1 if and only if  $I$  satisfies the sentence  $\chi$ . Both proofs are quite intricate, and the details of the syntax definition of  $\text{RGC}^k$  had to be tweaked right in order to enable proving *both* directions.

Plugging together the results achieved in the previous sections yields our main theorem, provided in Section 9: Two hypergraphs are homomorphism indistinguishable over the class  $GHW_k$  of hypergraphs of generalised hypertree width  $\leq k$  iff they are indistinguishable by the logic  $\text{RGC}^k$ .

Due to space limitations, many proof details had to be deferred to the paper’s extended version [22].

## 2 Preliminaries

This section provides basic notions concerning hypergraphs, incidence graphs, hypertree decompositions, and homomorphisms. We write  $\mathbb{R}$  for the set of reals,  $\mathbb{N}$  for the set of non-negative integers, and we let  $\mathbb{N}_{\geq 1} := \mathbb{N} \setminus \{0\}$  and  $[n] := \{1, \dots, n\}$  for all  $n \in \mathbb{N}_{\geq 1}$ .

**Hypergraphs.** The hypergraphs considered in this paper are generalisations of ordinary undirected graphs, where each edge can consist of an arbitrary number of vertices. For our proofs it will be necessary to deal with hypergraphs in which the same edge can have multiple occurrences. Furthermore, it will be convenient to assume that every vertex belongs to at least one edge. This is provided by the following definition that is basically taken from [7].

A *hypergraph*  $H := (V(H), E(H), f_H)$  consists of disjoint finite sets  $V(H)$  of *vertices* and  $E(H)$  of *edges*, and an *incidence function*  $f_H$  associating with every  $e \in E(H)$  the set  $f_H(e) \subseteq V(H)$  of vertices incident with edge  $e$ , such that  $V(H) = \bigcup_{e \in E(H)} f_H(e)$ . A *simple hypergraph* is a hypergraph  $H$  where the function  $f_H$  is injective. We can identify the edges of a hypergraph  $H$  with the multiset  $M_H := \{\{f_H(e) : e \in E(H)\}\}$ ; the number of occurrences of a set  $s \subseteq V(H)$  in this multiset then is the number of occurrences of “edge  $s$ ” in  $H$ . The *simple hypergraphs* are the hypergraphs in which every “edge  $s$ ” has only one occurrence.

Every hypergraph  $H = (V(H), E(H), f_H)$  can be represented by an ordinary, bipartite graph  $I_H$  in the following way: The vertices  $v \in V(H)$  occur as *red* nodes of  $I_H$ , i.e.,  $R(I_H) := V(H)$ . The edges  $e \in E(H)$  occur as *blue* nodes of  $I_H$ , i.e.,  $B(I_H) := E(H)$ . And there is an edge from each blue node  $e$  to all red nodes  $v \in f_H(e)$ . I.e.,  $E(I_H) := \{(e, v) \in B(I_H) \times R(I_H) : v \in f_H(e)\}$ . The condition  $V(H) = \bigcup_{e \in E(H)} f_H(e)$  implies that every red node is adjacent to at least one blue node. It is straightforward to see that the mapping  $H \mapsto I_H$  provides a bijection between the class of all hypergraphs and the class of all *incidence graphs*, where the notion of incidence graphs is as follows.

An *incidence graph*  $I = (R(I), B(I), E(I))$  consists of disjoint finite sets  $R(I)$  and  $B(I)$  of *red* nodes and *blue* nodes, resp., and a set of edges  $E(I) \subseteq B(I) \times R(I)$ , such that each red node is adjacent to at least one blue node. As usual for graphs, the *neighbourhood* of a node  $v$  is the set  $N_I(v)$  of all nodes adjacent to  $v$ . Thus, if  $I$  is the incidence graph  $I_H$  of a hypergraph  $H$ , the neighbourhood of every blue node  $e$  is  $N_I(e) = f_H(e)$ , i.e., the set of all vertices of  $H$  that are incident with edge  $e$ . The neighbourhood of every red node  $v$  is  $N_I(v) = \{e \in E(H) : v \in f_H(e)\}$ , i.e., the set of all edges of  $H$  that are incident with vertex  $v$ . Two incidence graphs  $I, I'$  are *isomorphic* ( $I \cong I'$ , for short) if there exists an *isomorphism*  $\pi = (\pi_R, \pi_B)$  from  $I$  to  $I'$ , i.e, bijections  $\pi_R : R(I) \rightarrow R(I')$  and  $\pi_B : B(I) \rightarrow B(I')$  such that for all  $(e, v) \in B(I) \times R(I)$  we have:  $(e, v) \in E(I) \iff (\pi_B(e), \pi_R(v)) \in E(I')$ . We sometimes drop the subscript and write  $\pi(e)$  and  $\pi(v)$  instead of  $\pi_B(e)$  and  $\pi_R(v)$ .

**Generalised Hypertree Decompositions.** We use the same notation as [12] for decompositions of hypergraphs, but we write  $bag(t)$  and  $cover(t)$  instead of  $\chi(t)$  and  $\lambda(t)$ , respectively, and we formalise them with respect to incidence graphs rather than hypergraphs.

► **Definition 2.1.** A *complete generalised hypertree decomposition* (*ghd*, for short) of an incidence graph  $I$  is a tuple  $D := (T, bag, cover)$ , where  $T := (V(T), E(T))$  is a finite undirected tree, and  $bag$  and  $cover$  are mappings that associate with every tree-node  $t \in V(T)$  a set  $bag(t) \subseteq R(I)$  of red nodes of  $I$  and a set  $cover(t) \subseteq B(I)$  of blue nodes of  $I$ , having the following properties:

1. *Completeness:* For each  $e \in B(I)$  there is a  $t \in V(T)$  with  $N_I(e) \subseteq bag(t)$  and  $e \in cover(t)$ .
2. *Connectedness for red nodes:* For every  $v \in R(I)$  the subgraph  $T_v$  of  $T$  induced on  $V_v := \{t \in V(T) : v \in bag(t)\}$  is a tree.
3. *Covering of Bags:* For every  $t \in V(T)$  we have  $bag(t) \subseteq \bigcup_{e \in cover(t)} N_I(e)$ .

It is straightforward to see that this notion of a *ghd* of an incidence graph  $I$  coincides with the classical notion (cf., [13, 12]) of a complete generalised hypertree decomposition of a hypergraph  $H$  where  $I_H = I$ . The *width*  $w(D)$  of a *ghd*  $D$  is defined as the maximum number of blue nodes in the cover of a tree-node, i.e.,  $w(D) := \max\{|cover(t)| : t \in V(T)\}$ . We write  $ghds(I)$  to denote the class of all *ghds* of an incidence graph  $I$ . The *generalised hypertree width* of an incidence graph  $I$  is  $ghw(I) := \min\{w(D) : D \in ghds(I)\}$ . By  $IGHW_k$  we denote the class of all incidence graphs of generalised hypertree width  $\leq k$ . It is straightforward to see that  $ghw(I_H)$  coincides with the classical notion (cf., [12]) of generalised hypertree width of a hypergraph  $H$ , and  $IGHW_k$  is the class of incidence graphs  $I_H$  of all hypergraphs  $H$  of generalised hypertree width  $\leq k$ .

For our proofs we need ghds with specific further properties, defined as follows; we are not aware of any related work that studies this particular kind of decompositions. Hypertree decompositions satisfying condition 4 (but not necessarily condition 5) of Definition 2.2 are known as *strong* decompositions [14].

► **Definition 2.2.** An *entangled hypertree decomposition* (ehd, for short) of an incidence graph  $I$  is a ghd  $D$  of  $I$  that additionally satisfies the following requirements:

4. *Precise coverage of bags:* For all tree-nodes  $t \in V(T)$  we have  $\bigcup_{e \in \text{cover}(t)} N_I(e) = \text{bag}(t)$ .
5. *Connectedness for blue nodes:* For every  $e \in B(I)$  the subgraph  $T_e$  of  $T$  induced on  $V_e := \{t \in V(T) : e \in \text{cover}(t)\}$  is a tree.

We write  $\text{ehds}(I)$  to denote the class of all ehds of an incidence graph  $I$ .

The *entangled hypertree width* of an incidence graph  $I$  is  $\text{ehw}(I) := \min\{w(D) : D \in \text{ehds}(I)\}$ . For a hypergraph  $H$  we let  $\text{ehw}(H) := \text{ehw}(I_H)$ . By  $\text{IEHW}_k$  we denote the class of all incidence graphs of entangled hypertree width  $\leq k$ .

Applying results from [2, 1, 3] shows that there exist arbitrarily large  $k$  such that  $\text{IEHW}_k$  is a strict subclass of  $\text{IGHW}_k$ . More precisely:

► **Theorem 2.3.**  $\text{IEHW}_k \subseteq \text{IGHW}_k$ , for every  $k \in \mathbb{N}_{\geq 1}$ . Furthermore,  $\text{IEHW}_1 = \text{IGHW}_1$ , but  $\text{IEHW}_k \subsetneq \text{IGHW}_k$  for each  $k \in \{2, 3\}$ . Moreover, for every  $n \in \mathbb{N}$  there exists a  $k \in \mathbb{N}_{\geq 1}$  such that  $\text{IGHW}_k \subsetneq \text{IEHW}_{k+n}$  (and hence,  $\text{IEHW}_{k+n} \subsetneq \text{IGHW}_{k+n}$ ).

**Proof.**  $\text{IEHW}_k \subseteq \text{IGHW}_k$  holds because every ehd also is a ghd.  $\text{IEHW}_1 = \text{IGHW}_1$  holds because ghds of width 1 are known to be equivalent to so-called *join trees*, and these can easily be translated into ehds of width 1. For the remaining statements, we use elaborate results from [2, 1, 3] that relate the *hypertree width*  $\text{hw}(H)$  (cf., [13, 14]) of a hypergraph to its *generalised hypertree width*  $\text{ghw}(H)$ :

From [2, Proposition 3.3.2] (cf. also [3, Example 3]) and [1, Claim 6.1] we obtain for each  $k \in \{2, 3\}$  a simple hypergraph  $H_k$  such that  $\text{ghw}(H_k) = k$  and  $\text{hw}(H_k) = k+1$ . Furthermore, [2, Fact 3.3.1] and [1, Theorem 4.1] provide for every  $n \in \mathbb{N}_{\geq 1}$  a simple hypergraph  $H^n$  such that  $\text{hw}(H^n) = \text{ghw}(H^n) + n$ .<sup>1</sup>

It is straightforward to verify that every ehd also is a complete *hypertree decomposition* in the sense of [13, 14]. Consequently, for every hypergraph  $H$  we have  $\text{hw}(H) \leq \text{ehw}(H)$ . Therefore, for each  $k \in \{2, 3\}$ , the incidence graph of  $H_k$  witnesses that  $\text{IEHW}_k \subsetneq \text{IGHW}_k$ .

To address the theorem's next statement, consider an arbitrary  $n \in \mathbb{N}$ . Let  $H := H^{n+1}$  and let  $k := \text{ghw}(H)$ . Then,  $\text{ehw}(H) \geq \text{hw}(H) = k+n+1$ . Thus, the incidence graph of  $H$  belongs to  $\text{IGHW}_k$  but not to  $\text{IEHW}_{k+n}$ . ◀

**Homomorphisms.** We use the classical notions for hypergraphs and incidence graphs:

A *homomorphism* from a hypergraph  $F$  to a hypergraph  $H$  is a pair  $(h_V, h_E)$  of mappings  $h_V : V(F) \rightarrow V(H)$  and  $h_E : E(F) \rightarrow E(H)$  such that for all  $e \in E(F)$  we have  $f_H(h_E(e)) = \{h_V(v) : v \in f_F(e)\}$ . We write  $\text{Hom}(F, H)$  for the set of all homomorphisms from  $F$  to  $H$ , and  $\text{hom}(F, H) := |\text{Hom}(F, H)|$  is the number of homomorphisms from  $F$  to  $H$ .

A *homomorphism* from an incidence graph  $J$  to an incidence graph  $I$  is a pair  $h = (h_R, h_B)$  of mappings  $h_R : R(J) \rightarrow R(I)$  and  $h_B : B(J) \rightarrow B(I)$  such that for all  $(e, v) \in E(J)$  we have  $(h_B(e), h_R(v)) \in E(I)$ . We sometimes drop the subscript and write  $h(e)$  and  $h(v)$  instead of  $h_B(e)$  and  $h_R(v)$ . By  $\text{Hom}(J, I)$  we denote the set of all homomorphisms from  $J$  to  $I$ , and we let  $\text{hom}(J, I) := |\text{Hom}(J, I)|$  be the number of homomorphisms from  $J$  to  $I$ .

<sup>1</sup> Note that the notions  $c_H\text{-hw}(H)$  and  $c_H\text{-ghw}(H)$  in [2] correspond to  $\text{hw}(H)$  and  $\text{ghw}(H)$  for all hypergraphs  $H$  according to [2, Example 2.1.10].

As pointed out in [7], every homomorphism from a hypergraph  $F$  to a hypergraph  $H$  also is a homomorphism from the incidence graph  $I_F$  to the incidence graph  $I_H$ ; but there exist homomorphisms from  $I_F$  to  $I_H$  that do not correspond to any homomorphism from  $F$  to  $H$ . In fact, every homomorphism  $(h_R, h_B)$  from  $I_F$  to  $I_H$  is a pair of mappings  $(h_V, h_E) := (h_R, h_B)$  with  $h_V : V(F) \rightarrow V(H)$  and  $h_E : E(F) \rightarrow E(H)$  such that for every  $e \in E(F)$  we have  $f_H(h_E(e)) \supseteq \{h_V(v) : v \in f_F(e)\}$  – i.e., the condition “=” of the definition of hypergraph-homomorphisms is relaxed into the condition “ $\supseteq$ ”.

### 3 Homomorphism Indistinguishability

Let  $(B, B', \mathcal{C})$  be either two incidence graphs and a class of incidence graphs or two hypergraphs and a class of hypergraphs. By  $\text{HOM}_{\mathcal{C}}(B)$  we denote the function  $\alpha : \mathcal{C} \rightarrow \mathbb{N}$  that associates with every  $A \in \mathcal{C}$  the number  $\text{hom}(A, B)$  of homomorphisms from  $A$  to  $B$ . We say that  $B$  and  $B'$  are *homomorphism indistinguishable over  $\mathcal{C}$*  if  $\text{HOM}_{\mathcal{C}}(B) = \text{HOM}_{\mathcal{C}}(B')$ . Note that  $\text{HOM}_{\mathcal{C}}(B) \neq \text{HOM}_{\mathcal{C}}(B')$  means that there exists an  $A \in \mathcal{C}$  that *distinguishes* between  $B$  and  $B'$  in the sense that  $\text{hom}(A, B) \neq \text{hom}(A, B')$ .

Recall from Section 2 that  $\text{IGHW}_k$  is the class of incidence graphs of generalised hypertree width  $\leq k$ . We write  $\text{GHW}_k$  for the class of all hypergraphs of generalised hypertree width  $\leq k$  (i.e., all hypergraphs  $H$  for which  $I_H \in \text{IGHW}_k$ ), and  $\text{sGHW}_k$  for the subclass consisting of all *simple* hypergraphs (i.e., hypergraphs where each edge has multiplicity 1) in  $\text{GHW}_k$ .

► **Theorem 3.1** (implicit in [7]). *Let  $H, H'$  be hypergraphs.*

(a) *If  $H$  and  $H'$  are simple hypergraphs, then*

$$\text{HOM}_{\text{GHW}_k}(H) = \text{HOM}_{\text{GHW}_k}(H') \iff \text{HOM}_{\text{sGHW}_k}(H) = \text{HOM}_{\text{sGHW}_k}(H').$$

(b)  $\text{HOM}_{\text{GHW}_k}(H) = \text{HOM}_{\text{GHW}_k}(H') \iff \text{HOM}_{\text{IGHW}_k}(I_H) = \text{HOM}_{\text{IGHW}_k}(I_{H'}).$

Böker [7] proved the analogous statement for  $BA, IBA$  instead of  $\text{GHW}_k, \text{IGHW}_k$ , where  $BA$  is the class of all Berge-acyclic hypergraphs and  $IBA$  is the class of all incidence graphs of hypergraphs in  $BA$ . Böker’s proof, however, works for all classes  $\mathcal{C}$  of hypergraphs and the associated class  $IC$  of all incidence graphs of hypergraphs in  $\mathcal{C}$ , provided that  $\mathcal{C}$  satisfies some mild closure properties, which  $\text{GHW}_k$  satisfies.

### 4 Relating $\text{IGHW}_k$ to $\text{IEHW}_k$

Recall from Section 2 that  $\text{IEHW}_k \subseteq \text{IGHW}_k$ , for the class  $\text{IEHW}_k$  of incidence graphs of entangled hypertree width  $\leq k$ . By Theorem 2.3 there exist arbitrarily large  $k$  such that  $\text{IEHW}_k$  is a strict subclass of  $\text{IGHW}_k$ . This section’s main result is that, nevertheless:

► **Theorem 4.1.** *For all incidence graphs  $I$  and  $I'$  we have*

$$\text{HOM}_{\text{IGHW}_k}(I) = \text{HOM}_{\text{IGHW}_k}(I') \iff \text{HOM}_{\text{IEHW}_k}(I) = \text{HOM}_{\text{IEHW}_k}(I').$$

**Proof sketch.** The proof heavily relies on our following technical main lemma, which uses the following notation: For an arbitrary incidence graph  $J$ , for  $s \subseteq R(J)$ , and for  $n \in \mathbb{N}$  we write  $J + n \cdot s$  to denote the incidence graph  $J'$  obtained from  $J$  by inserting  $n$  new blue nodes  $\hat{e}_1, \dots, \hat{e}_n$  and edges  $(\hat{e}_i, v)$  for all  $i \in [n]$  and all  $v \in s$  – i.e.,  $N_{J'}(\hat{e}_i) = s$ .

► **Lemma 4.2.** *Let  $J, I, I'$  be incidence graphs with  $\text{hom}(J, I) \neq \text{hom}(J, I')$ , let  $e \in B(J)$ , and let  $s \subseteq N_J(e)$ . For every  $m \in \mathbb{N}$  there exists an  $n \in \mathbb{N}$  with  $n \geq m$  such that  $J_n := J + n \cdot s$  satisfies  $\text{hom}(J_n, I) \neq \text{hom}(J_n, I')$ .*

The (combinatorially quite involved) proof of Lemma 4.2 can be found in the paper’s extended version [22].

The direction “ $\implies$ ” of Theorem 4.1 is trivial. For the direction “ $\impliedby$ ” it suffices to prove the following: If there is a  $J \in \text{IGHW}_k$  with  $\text{hom}(J, I) \neq \text{hom}(J, I')$ , then there also exists a  $J' \in \text{IEHW}_k$  with  $\text{hom}(J', I) \neq \text{hom}(J', I')$ . We construct such a  $J'$  in a 2-step process. We start with a ghd  $D = (T, \text{bag}, \text{cover})$  of  $J$  with  $w(D) \leq k$ . First, we transform  $D$  into a ghd  $D^1$  of an incidence graph  $J^1$  such that  $w(D^1) \leq w(D)$  and  $\text{hom}(J^1, I) \neq \text{hom}(J^1, I')$  and  $D^1$  satisfies condition 4 of Definition 2.2 (but condition 5 might still be violated). Afterwards, we transform  $D^1$  into a ghd  $D^2$  of an incidence graph  $J^2$  such that  $w(D^2) = w(D^1)$  and  $\text{hom}(J^2, I) \neq \text{hom}(J^2, I')$  and  $D^2$  satisfies conditions 4 and 5 of Definition 2.2 and hence is an ehd. Letting  $J' := J^2$  then completes the proof.

For the construction of  $D^1, J^1$  we consider all those  $t \in V(T)$  and  $e \in \text{cover}(t)$  where  $N_J(e) \not\subseteq \text{bag}(t)$  and let  $s := N_J(e) \cap \text{bag}(t)$ . We use Lemma 4.2 to choose a suitable number  $n_s \geq 1$  and replace  $J$  by  $J + n_s \cdot s$  (let us write  $e'_1, \dots, e'_{n_s}$  for the  $n_s$  newly inserted blue nodes). In  $D$  we replace  $e$  with  $e'_1$  in  $\text{cover}(t)$ , and we add new leaves  $t_j$  for  $j \in \{2, \dots, n_s\}$  adjacent to  $t$  with  $\text{cover}(t_j) = \{e'_j\}$  and  $\text{bag}(t_j) = s$ . After having done this for all combinations of  $t$  and  $e$ , we end up with the desired incidence graph  $J^1$  and ghd  $D^1 = (T^1, \text{bag}^1, \text{cover}^1)$ .

For the construction of  $D^2, J^2$ , for each  $e \in B(J^1)$  we let  $m_e$  be the number of connected components of the subgraph  $T_e^1$ , i.e., the subgraph of  $T^1$  induced on  $V_e := \{t \in V(T^1) : e \in \text{cover}^1(t)\}$ . Let  $V_{e,0}, \dots, V_{e,m_e-1}$  be the sets of tree-nodes (i.e., nodes in  $V(T^1)$ ) of these connected components. We consider all those  $e \in B(J^1)$  where  $m_e \geq 2$  and let  $s := N_{J^1}(e)$ . We use Lemma 4.2 to choose a suitable number  $n_e \geq m_e - 1$  and replace  $J$  with  $J + n_e \cdot s$  (let us write  $e'_1, \dots, e'_{n_e}$  for the  $n_e$  newly inserted blue nodes). In  $D^1$  we consider for every  $i \in \{1, \dots, m_e - 1\}$  all  $t \in V_{e,i}$  and replace  $e$  with  $e'_i$  in  $\text{cover}^1(t)$ . Furthermore, we pick an arbitrary  $t \in V_{e,0}$ , and for each  $i \in [n_e]$  with  $i \geq m_e$ , we insert into  $T^1$  a new leaf  $t_{e,i}$  adjacent to  $t$  and let  $\text{bag}^1(t_{e,i}) := s$  and  $\text{cover}^1(t_{e,i}) := \{e'_{e,i}\}$ . After having done this for all  $e \in B(J^1)$  with  $m_e \geq 2$ , we end up with the desired incidence graph  $J^2$  and ehd  $D^2$ . This completes the proof sketch of Theorem 4.1.  $\blacktriangleleft$

## 5 Notation for Partial Functions

We introduce some further notation that will be convenient for the remaining parts of the paper. We write  $f : A \rightarrow B$  to indicate that  $f$  is a partial function from  $A$  to  $B$ . By  $\text{dom}(f)$  we denote the domain of  $f$ , i.e., the set of all  $a \in A$  on which  $f(a)$  is defined. By  $\text{img}(f)$  we denote the image of  $f$ , i.e.,  $\text{img}(f) = \{f(a) : a \in \text{dom}(f)\}$ . Two partial functions  $f : A \rightarrow B$  and  $g : A \rightarrow B$  are called *compatible* if  $f(a) = g(a)$  holds for all  $a \in \text{dom}(f) \cap \text{dom}(g)$ .

We identify a partial function  $f$  with the set  $\{(a, f(a)) : a \in \text{dom}(f)\}$ . This allows us to compare and combine partial functions via standard notation from set theory. E.g.,  $f \subseteq g$  indicates that  $\text{dom}(f) \subseteq \text{dom}(g)$  and  $f(a) = g(a)$  for all  $a \in \text{dom}(f)$ . And  $f \cup g$  denotes the partial function  $h$  with  $\text{dom}(h) = \text{dom}(f) \cup \text{dom}(g)$  and  $h(a) = f(a)$  for all  $a \in \text{dom}(f)$  and  $h(a) = g(a)$  for all  $a \in \text{dom}(g) \setminus \text{dom}(f)$ ; note that  $f$  has precedence over  $g$  in case that  $f$  and  $g$  are not compatible. For a set  $S$  we write  $f - S$  to denote the partial function  $g$  with  $g \subseteq f$  and  $\text{dom}(g) = \text{dom}(f) \setminus S$ .

## 6 2-Sorted Counting Logic with Guards: $\text{GC}^k$ and $\text{RGC}^k$

This section provides the syntax and semantics of our 2-sorted logics  $\text{GC}^k$  and  $\text{RGC}^k$ . Formulas of these logics are evaluated on incidence graphs (cf. Section 2). We fix a  $k \in \mathbb{N}_{\geq 1}$ .

To address *blue* nodes (i.e., *edges* of a hypergraph), we have available  $k$  *blue variables*  $\mathbf{e}_1, \dots, \mathbf{e}_k$ . To address *red* nodes (i.e., *vertices* of a hypergraph), we have available countably many *red variables*  $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots$ . An atomic formula  $E(\mathbf{e}_j, \mathbf{v}_i)$  states that a hypergraph's

vertex  $v_i$  is included in the hypergraph's edge  $e_j$ . Let  $\text{Var}_B := \{e_1, \dots, e_k\}$ ,  $\text{Var}_R := \{v_i : i \in \mathbb{N}_{\geq 1}\}$ , and  $\text{Var} := \text{Var}_B \cup \text{Var}_R$ . An *interpretation*  $\mathcal{I} = (I, \beta)$  consists of an incidence graph  $I = (R(I), B(I), E(I))$  and an *assignment*  $\beta$  in  $I$ , i.e., a mapping  $\beta : \text{Var} \rightarrow R(I) \cup B(I)$  with  $\beta(e_j) \in B(I)$  for all  $e_j \in \text{Var}_B$  and  $\beta(v_i) \in R(I)$  for all  $v_i \in \text{Var}_R$ . In the formulas of our logics, red variables  $v_i$  have to be *guarded* by a blue variable  $e_j$  in the sense that  $E(e_j, v_i)$  holds. This is formalised by a *guard function*, i.e., a *partial function*  $g : \mathbb{N}_{\geq 1} \rightarrow [k]$  with *finite* domain  $\text{dom}(g)$ . Every guard function  $g$  corresponds to the formula  $\Delta_g := \bigwedge_{i \in \text{dom}(g)} E(e_{g(i)}, v_i)$ , and for the special case where  $\text{dom}(g) = \emptyset$  we let  $\Delta_g := \top$  where  $\top$  is a special atomic formula satisfied by *every* interpretation  $\mathcal{I}$ . We let  $\text{free}(\Delta_g)$  be the set of all (red or blue) variables that occur in  $\Delta_g$ . An interpretation  $\mathcal{I} = (I, \beta)$  *satisfies* a guard function  $g$  (in symbols:  $\mathcal{I} \models \Delta_g$ ) if for all  $i \in \text{dom}(g)$  we have:  $(\beta(e_{g(i)}), \beta(v_i)) \in E(I)$ . I.e., for every  $i \in \text{dom}(g)$ , the red variable  $v_i$  is guarded by the blue variable  $e_{g(i)}$  in the sense that it is connected to it by an edge of the incidence graph.

For any formula  $\chi$  we write  $\text{ifree}_B(\chi)$  for the set of all indices  $j \in [k]$  such that the blue variable  $e_j$  belongs to  $\text{free}(\chi)$ . Accordingly,  $\text{ifree}_R(\chi) := \{i \in \mathbb{N}_{\geq 1} : v_i \in \text{free}(\chi)\}$ . The definition of the syntax of  $\text{GC}^k$  is inductively given as follows.

**Base cases:** The atomic formulas in  $\text{GC}^k$  are of the form  $\top$ ,  $E(e_j, v_i)$ ,  $e_j = e_{j'}$ , and  $v_i = v_{i'}$  for  $j, j' \in [k]$  and  $i, i' \in \mathbb{N}_{\geq 1}$ .

**Inductive cases:**

1. If  $\psi \in \text{GC}^k$ , then  $\neg\psi \in \text{GC}^k$ .
2. If  $\psi_1, \psi_2 \in \text{GC}^k$ , then  $(\psi_1 \wedge \psi_2) \in \text{GC}^k$ .
3. If  $\psi \in \text{GC}^k$  and  $g$  is a guard function with  $\text{dom}(g) = \text{ifree}_R(\psi)$  and  $n, \ell \in \mathbb{N}_{\geq 1}$  and, for  $\chi := (\Delta_g \wedge \psi)$  and  $i_1 < \dots < i_\ell$  with
  - (a)  $i_1, \dots, i_\ell \in \text{ifree}_R(\chi)$ , then  $\varphi \in \text{GC}^k$  for  $\varphi := \exists^{\geq n}(v_{i_1}, \dots, v_{i_\ell}).(\Delta_g \wedge \psi)$ ;
  - (b)  $i_1, \dots, i_\ell \in \text{ifree}_B(\chi)$ , then  $\varphi \in \text{GC}^k$  for  $\varphi := \exists^{\geq n}(e_{i_1}, \dots, e_{i_\ell}).(\Delta_g \wedge \psi)$ .

The semantics are defined as expected. In particular, an interpretation  $\mathcal{I} = (I, \beta)$  satisfies the formula  $\varphi := \exists^{\geq n}(v_{i_1}, \dots, v_{i_\ell}).(\Delta_g \wedge \psi)$  iff there are at least  $n$  tuples  $(v_{i_1}, \dots, v_{i_\ell}) \in R(I)^\ell$  such that  $\mathcal{I}' = (I, \beta')$  satisfies  $(\Delta_g \wedge \psi)$ , where  $\beta'(v_{i_j}) = v_{i_j}$  for all  $j \in [\ell]$  and  $\beta'(\mathbf{x}) = \beta(\mathbf{x})$  for all  $\mathbf{x} \in \text{Var} \setminus \{v_{i_1}, \dots, v_{i_\ell}\}$ . Similarly,  $\mathcal{I} = (I, \beta)$  satisfies  $\varphi := \exists^{\geq n}(e_{i_1}, \dots, e_{i_\ell}).(\Delta_g \wedge \psi)$  iff there are at least  $n$  tuples  $(e_{i_1}, \dots, e_{i_\ell}) \in B(I)^\ell$  such that  $\mathcal{I}' = (I, \beta')$  satisfies  $(\Delta_g \wedge \psi)$ , where  $\beta'(e_{i_j}) = e_{i_j}$  for all  $j \in [\ell]$  and  $\beta'(\mathbf{x}) = \beta(\mathbf{x})$  for all  $\mathbf{x} \in \text{Var} \setminus \{e_{i_1}, \dots, e_{i_\ell}\}$ . Obviously we can emulate the  $\forall$ -quantifier (and disjunction) using  $\exists^{\geq 1}$  and  $\neg$  (and  $\wedge$  and  $\neg$ , respectively).

We write  $\mathcal{I} \models \chi$  to indicate that  $\mathcal{I}$  satisfies the formula  $\chi$ ; and  $\mathcal{I} \not\models \chi$  indicates that  $\mathcal{I}$  does not satisfy  $\chi$ . *Sentences* of  $\text{GC}^k$  are formulas  $\chi \in \text{GC}^k$  with  $\text{free}(\chi) = \emptyset$ . For an incidence graph  $I$  and a sentence  $\chi \in \text{GC}^k$  we write  $I \models \chi$  to indicate that  $\mathcal{I} \models \chi$  where  $\mathcal{I} = (I, \beta)$  for any assignment  $\beta$  in  $I$  (since  $\chi$  has no free variable, the assignment does not matter). For a hypergraph  $H$  and a sentence  $\chi \in \text{GC}^k$  we write  $H \models \chi$  to indicate that  $I_H \models \chi$ . For two incidence graphs  $I$  and  $I'$  we write  $I \equiv_{\text{GC}^k} I'$  and say that  $I$  and  $I'$  are *indistinguishable by the logic*  $\text{GC}^k$  if for all sentences  $\chi \in \text{GC}^k$  we have:  $I \models \chi \iff I' \models \chi$ .

Let us now introduce a restriction of  $\text{GC}^k$  that we call  $\text{RGC}^k$ . Every formula of  $\text{RGC}^k$  will be of the form  $(\Delta_g \wedge \psi)$ , where  $g$  is a guard function whose domain  $\text{dom}(g)$  consists of all indices  $i \in \mathbb{N}_{\geq 1}$  such that the red variable  $v_i$  is a free variable of  $\psi$ . We let  $\text{free}((\Delta_g \wedge \psi)) := \text{free}(\Delta_g) \cup \text{free}(\psi)$  denote the set of free variables of the formula. The definition of the syntax of  $\text{RGC}^k$  is inductively given as follows.

**Base cases:**  $(\Delta_g \wedge \psi) \in \text{RGC}^k$  for all  $\psi$  and all  $g : \mathbb{N}_{\geq 1} \rightarrow [k]$  matching one of the following:

1.  $\psi$  is  $E(e_j, v_i)$  and  $\text{dom}(g) = \{i\}$  and  $j \in [k]$  (note that  $g(i) \in [k]$  can be chosen arbitrarily);
2.  $\psi$  is  $e_j = e_{j'}$  with  $\text{dom}(g) = \emptyset$  and  $j, j' \in [k]$ ;
3.  $\psi$  is  $v_i = v_{i'}$  with  $\text{dom}(g) = \{i, i'\}$ .

**Inductive cases:**

4. If  $(\Delta_g \wedge \psi) \in \text{RGC}^k$ , then  $(\Delta_g \wedge \neg\psi) \in \text{RGC}^k$ ;
5. If  $(\Delta_{g_i} \wedge \psi_i) \in \text{RGC}^k$  for  $i \in [2]$  and  $g_1$  and  $g_2$  are compatible (i.e., they agree on  $\text{dom}(g_1) \cap \text{dom}(g_2)$ ), then  $(\Delta_g \wedge \varphi) \in \text{RGC}^k$  for  $g := g_1 \cup g_2$  and  $\varphi := (\psi_1 \wedge \psi_2)$ ;
6. If  $(\Delta_g \wedge \psi) \in \text{RGC}^k$  and  $n, \ell \in \mathbb{N}_{\geq 1}$ , and  $i_1, \dots, i_\ell \in \text{dom}(g)$  with  $i_1 < \dots < i_\ell$ , then  $(\Delta_{\tilde{g}} \wedge \varphi) \in \text{RGC}^k$  for  $\varphi := \exists^{\geq n}(\mathbf{v}_{i_1}, \dots, \mathbf{v}_{i_\ell}).(\Delta_g \wedge \psi)$  and  $\tilde{g} := g - \{i_1, \dots, i_\ell\}$  (note that  $\text{free}(\varphi) = \text{free}((\Delta_g \wedge \psi)) \setminus \{\mathbf{v}_{i_1}, \dots, \mathbf{v}_{i_\ell}\}$ );
7. If  $(\Delta_g \wedge \psi) \in \text{RGC}^k$  and  $n, \ell \in \mathbb{N}_{\geq 1}$ , and  $S := \{i_1, \dots, i_\ell\} \subseteq \text{ifree}_B(\chi)$  for  $\chi := (\Delta_g \wedge \psi)$  with  $i_1 < \dots < i_\ell$ , and if  $\tilde{g} : \mathbb{N}_{\geq 1} \rightarrow [k]$  with  $\text{dom}(\tilde{g}) = \text{dom}(g)$  such that all  $i \in \text{dom}(g)$  satisfy

$$\tilde{g}(i) = g(i) \quad \text{or} \quad \tilde{g}(i) \in S \quad \text{or} \quad \tilde{g}(i) \notin \text{img}(g), \quad \text{then} \quad (1)$$

$$(\Delta_{\tilde{g}} \wedge \varphi) \in \text{RGC}^k \text{ for } \varphi := \exists^{\geq n}(\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_\ell}).(\Delta_g \wedge \psi) \text{ (here, } \text{free}(\varphi) = \text{free}(\chi) \setminus \{\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_\ell}\}).$$

Let us have a closer look at rule 7): The formula  $\varphi$  has exactly the same free red variables as the formula  $\chi$ . But the guard of red variable  $\mathbf{v}_i$  in  $\tilde{\chi} := (\Delta_{\tilde{g}} \wedge \varphi)$  is  $j' := \tilde{g}(i)$ , whereas in  $\chi$  it is  $j := g(i)$ . Condition (1) is equivalent to the following: the guard remains unchanged (i.e.,  $j' = j$ ), or the new guard  $j'$  has become “available” by the quantification (i.e.,  $j' \in S$ ) or it has not been used as a guard by  $g$  (i.e.,  $j' \notin \text{img}(g)$ ).

Note that  $\text{RGC}^k \subseteq \text{GC}^k$ . Furthermore, for all  $\chi := (\Delta_g \wedge \psi) \in \text{RGC}^k$  we have  $\text{dom}(g) = \{i \in \mathbb{N}_{\geq 1} : \mathbf{v}_i \in \text{free}(\chi)\}$ . Sentences of  $\text{RGC}^k$  are formulas  $\chi := (\Delta_g \wedge \psi)$  in  $\text{RGC}^k$  with  $\text{free}(\chi) = \emptyset$ . Since  $\text{dom}(g) = \{i \in \mathbb{N}_{\geq 1} : \mathbf{v}_i \in \text{free}(\chi)\}$ , this implies that  $\text{dom}(g) = \emptyset$ , i.e.,  $g = g_\emptyset$  where  $g_\emptyset$  is the uniquely defined partial mapping with empty domain; recall that  $\Delta_{g_\emptyset} = \top$ . For two incidence graphs  $I$  and  $I'$  we write  $I \equiv_{\text{RGC}^k} I'$  and say that  $I$  and  $I'$  are *indistinguishable by the logic  $\text{RGC}^k$*  if for all sentences  $\chi \in \text{RGC}^k$  we have:  $I \models \chi \iff I' \models \chi$ . The subsequent sections of this paper are devoted to proving the following theorem, stating that indistinguishability by the logic  $\text{RGC}^k$  coincides with homomorphism indistinguishability over the class  $\text{IEHW}_k$  of incidence graphs of entangled hypertree width at most  $k$ .

► **Theorem 6.1.** *For all incidence graphs  $I, I'$  and all  $k \in \mathbb{N}_{\geq 1}$  we have:*

$$I \equiv_{\text{RGC}^k} I' \iff \text{HOM}_{\text{IEHW}_k}(I) = \text{HOM}_{\text{IEHW}_k}(I').$$

This result can be viewed as a lifting of Dvořák’s theorem [11] stating that any two graphs  $G, G'$  are indistinguishable by the  $k+1$ -variable logic  $C^{k+1}$  if, and only if, they are homomorphism indistinguishable over the class  $\text{TW}_k$  of graphs of tree-width  $\leq k$ . Our proof of Theorem 6.1 is heavily inspired by Dvořák’s proof. But in order to proceed along a similar construction, we first have to provide a suitable inductive characterisation of the class  $\text{IEHW}_k$ . This is presented in Section 7, where we also provide the machinery of *quantum* incidence graphs as an analogue of the quantum graphs used in Dvořák’s proof. Section 8 is devoted to the proof of Theorem 6.1.

Before we close this section, let us have a look at some examples. Let  $k = 2$ . Consider the following formula  $\psi_1 \in \text{GC}^k$ :

$$\psi_1 := \exists^{\geq 1}(\mathbf{v}_1).(E(\mathbf{e}_1, \mathbf{v}_1) \wedge E(\mathbf{e}_2, \mathbf{v}_1)).$$

$\psi_1$  expresses that the hyperedges  $\mathbf{e}_1$  and  $\mathbf{e}_2$  share at least one vertex  $\mathbf{v}_1$ , i.e. they intersect. Since we quantify over  $\mathbf{v}_1$ , the definition of  $\text{GC}^k$  requires us to insert a guard ranging over the set of free red variables, i.e. over  $\{\mathbf{v}_1\}$ . We chose  $E(\mathbf{e}_1, \mathbf{v}_1)$  as the guard but note that  $E(\mathbf{e}_2, \mathbf{v}_1)$  would have been a valid choice as well. Next, consider the formula  $\psi_2 \in \text{GC}^k$ :

$$\psi_2 := \bigwedge_{j \in \{1,2\}} \exists^{\geq 3}(\mathbf{v}_1).(E(\mathbf{e}_j, \mathbf{v}_1) \wedge E(\mathbf{e}_j, \mathbf{v}_1)).$$

$\psi_2$  expresses that each of the hyperedges  $e_1, e_2$  contains at least three vertices. Again, we have to insert a guard after the quantifier, which is why  $E(e_j, v_1)$  appears twice in  $\psi_2$  – as a guard *and* as our “actual” subformula.

Finally, we use the formulas  $\psi_1, \psi_2$  to construct a sentence  $\varphi \in \text{GC}^k$ :

$$\varphi := \neg \exists^{\geq 1} (e_1, e_2). (\top \wedge ((\psi_1 \wedge \psi_2) \wedge \neg e_1 = e_2)).$$

$\varphi$  expresses that there is no pair of non-equal hyperedges  $(e_1, e_2)$  that intersect and that both contain at least 3 vertices. I.e.,  $\varphi$  expresses that all hyperedges that contain at least 3 vertices are pairwise disjoint. Once again, the quantification requires us to insert a guard; since there are no free red variables, we insert  $\top$  as the guard. Note that  $(\top \wedge \varphi) \in \text{RGC}^k$ . For more examples of formulas in  $\text{GC}^k$  and  $\text{RGC}^k$ , consult the paper’s extended version [22].

## 7 An Inductive Characterisation of $\text{IEHW}_k$

In this section we give an inductive definition of what we call *guarded  $k$ -labeled incidence graphs* ( $\text{GLI}_k$ ), and we show that these are equivalent to the incidence graphs in  $\text{IEHW}_k$ . Throughout this section, we fix an arbitrary number  $k \in \mathbb{N}_{\geq 1}$ .

**$k$ -Labeled Incidence Graphs and the Class  $\text{GLI}_k$ .** We enrich an incidence graph  $I$  by labeling some of its blue (red) nodes with labels in  $[k]$  (in  $\mathbb{N}_{\geq 1}$ ), and by providing, for every  $i \in \mathbb{N}_{\geq 1}$  that is used as a label for a red node, a “blue label”  $g(i) \in [k]$  that should be regarded as “the guard” of  $i$ . Each label can only be used once, not all labels have to be used, not all vertices have to be labeled, one vertex may have multiple labels, and “guards” can be chosen arbitrarily. This is formalised as follows: A  *$k$ -labeled incidence graph*  $L = (I, r, b, g)$  consists of an incidence graph  $I$  and partial mappings  $r : \mathbb{N}_{\geq 1} \rightarrow R(I)$ ,  $b : [k] \rightarrow B(I)$ , and  $g : \mathbb{N}_{\geq 1} \rightarrow [k]$  such that  $\text{dom}(g) = \text{dom}(r)$  is finite. We write  $I_L, r_L, b_L, g_L$  to address  $L$ ’s components  $I, r, b, g$ . Let  $L = (I, r, b, g)$  be a  $k$ -labeled incidence graph. If  $j \in \text{dom}(b)$ , the blue node  $b(j)$  of  $I$  is labeled with the number  $j$ . If  $i \in \text{dom}(r)$ , the red node  $r(i)$  of  $I$  is labeled with the number  $i$ , and  $g(i) = j$  indicates that the blue node labeled with the number  $j$  (if it exists) should be regarded as “the guard” of the red node labeled with the number  $i$ .

We say that  $L$  *has real guards* if for every  $i \in \text{dom}(r)$  the red node  $v$  labeled  $i$  is “guarded” by the blue node  $e$  labeled  $j := g(i)$  in the sense that  $I$  contains an edge from  $e$  to  $v$ . This is formalised as follows: A  $k$ -labeled incidence graph  $L = (I, r, b, g)$  is said to *have real guards w.r.t.  $f$*  for a partial function  $f : \mathbb{N}_{\geq 1} \rightarrow [k]$  if  $\text{dom}(f) \subseteq \text{dom}(r)$  and for all  $i \in \text{dom}(f)$  we have  $f(i) \in \text{dom}(b)$  and  $(b(f(i)), r(i)) \in E(I)$ . We say that  $L$  *has real guards* if it has real guards w.r.t.  $g$ . Particularly simple examples of  $k$ -labeled incidence graphs with real guards are provided by the following definition.

► **Definition 7.1.** Let  $f : \mathbb{N}_{\geq 1} \rightarrow [k]$  with finite  $\text{dom}(f) \neq \emptyset$ . The  $k$ -labeled incidence graph  $M_f$  defined by  $f$  is the  $k$ -labeled incidence graph  $L = (I, r, b, g)$  with  $g := f$ , where  $I$  consists of a red node  $v_i$  for every  $i \in \text{dom}(f)$ , a blue node  $e_j$  for every  $j \in \text{img}(f)$ , and an edge  $(e_{f(i)}, v_i)$  for every  $i \in \text{dom}(f)$ , and where  $\text{dom}(r) = \text{dom}(f)$  and  $r(i) = v_i$  for all  $i \in \text{dom}(r)$ , and  $\text{dom}(b) = \text{img}(f)$  and  $b(j) = e_j$  for all  $j \in \text{dom}(b)$ . Note that  $M_f$  has real guards.

We introduce a number of operations on  $k$ -labeled incidence graphs. The first kind of operations provides ways to modify the labels (the latter two of these do not necessarily preserve real guards). Let  $L = (I, r, b, g)$  be a  $k$ -labeled incidence graph. Let  $X_r \subseteq \mathbb{N}_{\geq 1}$  be finite, and let  $X_b \subseteq [k]$ .

1. Removing from the red nodes all the labels in  $X_r$  is achieved by the operation  $L[X_r \rightarrow \bullet] := (I, r', b, g')$  with  $r' := r - X_r$  and  $g' := g - X_r$ .
2. Removing from the blue nodes all the labels in  $X_b$  is achieved by the operation  $L\langle X_b \rightarrow \bullet \rangle := (I, r, b', g)$  with  $b' := b - X_b$ .
3. Let  $X_r = \{i_1, \dots, i_\ell\}$  for  $\ell := |X_r|$  and  $i_1 < \dots < i_\ell$ . For every  $\bar{v} = (v_1, \dots, v_\ell) \in R(I)^\ell$  we let  $L[X_r \rightarrow \bar{v}] := (I, r', b, g)$  with  $\text{dom}(r') = \text{dom}(r) \cup X_r$  and  $r'(i_j) = v_j$  for all  $j \in [\ell]$  and  $r'(i) = r(i)$  for all  $i \in \text{dom}(r) \setminus X_r$  (i.e., for each  $j \in [\ell]$ , the red label  $i_j$  is moved onto the red node  $v_j$ , and all other labels remain unchanged).
4. Let  $X_b = \{i_1, \dots, i_\ell\}$  for  $\ell := |X_b|$  and  $i_1 < \dots < i_\ell$ . For every  $\bar{e} = (e_1, \dots, e_\ell) \in B(I)^\ell$  we let  $L\langle X_b \rightarrow \bar{e} \rangle := (I, r, b', g)$  with  $\text{dom}(b') = \text{dom}(b) \cup X_b$  and  $b'(i_j) = e_j$  for all  $j \in [\ell]$  and  $b'(i) = b(i)$  for all  $i \in \text{dom}(b) \setminus X_b$  (i.e., for each  $j \in [\ell]$ , the blue label  $i_j$  is moved onto the blue node  $e_j$ , and all other labels remain unchanged).

The next operation enables us to *glue* two  $k$ -labeled incidence graphs  $L_1$  and  $L_2$ . This is achieved by first taking the disjoint union of  $L_1$  and  $L_2$  and then merging all red (blue) nodes that carry the same label into a single red (blue) node that inherits all neighbours of the merged nodes. We write  $(L_1 \cdot L_2)$  to denote the resulting  $k$ -labeled incidence graph. We need one further operation on  $k$ -labeled incidence graphs, namely, one that admits us to change its guard function:

► **Definition 7.2** (Applying a transition). Consider a partial function  $g : \mathbb{N}_{\geq 1} \rightarrow [k]$ .

- (a) A *transition for  $g$*  is a partial function  $f : \mathbb{N}_{\geq 1} \rightarrow [k]$  with  $\emptyset \neq \text{dom}(f) \subseteq \text{dom}(g)$  satisfying the following: for every  $i \in \text{dom}(g)$  with  $g(i) \in \text{img}(f)$  we have  $i \in \text{dom}(f)$ .
- (b) Let  $L = (I, r, b, g)$  be a  $k$ -labeled incidence graph, and let  $f$  be a transition for  $g$ . *Applying* the transition  $f$  to  $L$  yields the  $k$ -labeled incidence graph  $L[\rightsquigarrow f] := (M_f \cdot L\langle X_b \rightarrow \bullet \rangle)$ , where  $X_b := \text{img}(g) \cap \text{img}(f) \cap \text{dom}(b)$ , and  $M_f$  is provided by Definition 7.1.

The idea of applying a transition  $f$  to a  $k$ -labeled incidence graph  $L = (I, r, b, g)$  is to assign new guards to a set of labeled red vertices (i.e. the domain of  $f$ ). These new guards should be newly inserted nodes, and they should be *real* guards. To this end, for every  $j \in \text{img}(f)$  we add a new blue node  $e'_j$  labeled  $j$ ; and in case that the label  $j$  had already been used by a blue node  $e$  of  $L$  (i.e.,  $j \in \text{dom}(b)$ ) and served as a guard according to  $g$  (i.e.,  $j \in \text{img}(g)$ ), we remove this label from  $e$ . For each  $i \in \text{dom}(f)$  with  $f(i) = j$  we add an edge from the red node of  $L$  labeled  $i$  to the new blue node  $e'_j$ .

The formal definition  $L[\rightsquigarrow f] := (M_f \cdot L\langle X_b \rightarrow \bullet \rangle)$  achieves this as follows: By  $L\langle X_b \rightarrow \bullet \rangle$  we release from  $L$  all blue labels  $j$  that are present in  $L$  and that we want to assign to newly created nodes. This is achieved by letting  $X_b = \text{img}(g) \cap \text{img}(f) \cap \text{dom}(b)$ . Afterwards, adding the edges from the nodes of  $L$  that carry a red label  $i \in \text{dom}(f)$  to the new blue node  $e'_{f(i)}$  is achieved by glueing  $M_f$  to  $L\langle X_b \rightarrow \bullet \rangle$ . Note that releasing from  $L$  all blue labels in  $X_b$  might be problematic: Consider a red node  $v$  labeled  $i$  that was originally guarded by the blue node  $e$  of  $L$  that carried the label  $j := g(i)$ . Releasing the label  $j$  from node  $e$  means that  $v$  loses its guard in case that  $i \notin \text{dom}(f)$ . Therefore, for  $f$  to be a transition for  $g$ , we require in Definition 7.2 that it assigns a new guard to all the affected labeled red vertices, i.e. we require  $i \in \text{dom}(f)$ , if  $g(i) \in \text{img}(f)$ . Finally, we are ready to define the class  $GLI_k$ :

► **Definition 7.3** ( $GLI_k$ ). The class  $GLI_k$  of *guarded  $k$ -labeled incidence graphs* is inductively defined as follows:

**Base case:** Any  $k$ -labeled incidence graph  $L = (I, r, b, g)$  with  $R(I) = \text{img}(r)$ ,  $B(I) = \text{img}(b)$ , and with real guards belongs to  $GLI_k$ .

**Inductive cases:** Let  $L = (I, r, b, g) \in GLL_k$ .

1.  $L[X_r \rightarrow \bullet] \in GLL_k$  for every  $X_r \subseteq \text{dom}(r)$ .
2.  $L\langle X_b \rightarrow \bullet \rangle \in GLL_k$  for every  $X_b \subseteq \text{dom}(b) \setminus \text{img}(g)$ .
3.  $L[\rightsquigarrow f] \in GLL_k$  for every transition  $f$  for  $g$ .
4.  $(L \cdot L') \in GLL_k$  for every  $L' = (I', r', b', g') \in GLL_k$  such that  $g$  and  $g'$  are compatible.

An easy inductive proof shows that every  $L \in GLL_k$  has real guards.

A  $k$ -labeled incidence graph  $L$  is called *label-free* if  $\text{dom}(r_L) = \text{dom}(b_L) = \text{dom}(g_L) = \emptyset$ . We are now ready for this section's technical main result, which states that, essentially,  $GLL_k$  provides an inductive characterisation of  $IEHW_k$ :

► **Theorem 7.4.**

- (a) *The incidence graph  $I_L$  of every  $L \in GLL_k$  is in  $IEHW_k$ .*
- (b) *For every  $I \in IEHW_k$  there exists a label-free  $L \in GLL_k$  such that  $I \cong I_L$ .*

The proof of (a) proceeds by induction on the definition of  $GLL_k$  and explicitly constructs an ehd of width  $\leq k$  for each  $L \in GLL_k$ ; for this it utilises that  $L$  has real guards. The proof of (b) starts with an ehd of  $I$  of width  $\leq k$ , chooses a suitable root node of the ehd's tree and performs a bottom-up traversal of this tree to associate each tree-node  $t$  with a corresponding  $k$ -labeled incidence graph  $L_t$ . This construction's details have to be carried out with care to ensure that  $L_t \in GLL_k$ .

**Homomorphisms on  $k$ -Labeled Incidence Graphs and their Quantum Analogues.**

We define the notion of *homomorphisms* of  $k$ -labeled incidence graphs in such a way that it respects labels, but ignores the guard function: Let  $L = (I, r, b, g)$  and  $L' = (I', r', b', g')$  be  $k$ -labeled incidence graphs. If  $\text{dom}(r) \not\subseteq \text{dom}(r')$  or  $\text{dom}(b) \not\subseteq \text{dom}(b')$ , then there exists no homomorphism from  $L$  to  $L'$ . Otherwise, a homomorphism from  $L$  to  $L'$  is a homomorphism  $h = (h_R, h_B)$  from  $I$  to  $I'$  satisfying the following condition:  $h(r(i)) = r'(i)$  for all  $i \in \text{dom}(r)$  and  $h(b(j)) = b'(j)$  for all  $j \in \text{dom}(b)$ . By  $\text{Hom}(L, L')$  we denote the set of all homomorphisms from  $L$  to  $L'$ , and we let  $\text{hom}(L, L') := |\text{Hom}(L, L')|$  be the number of homomorphisms from  $L$  to  $L'$ . In particular, if  $L$  is *label-free*, then  $\text{hom}(L, L') = \text{hom}(I_L, I_{L'})$ .

In order to enable us to “aggregate” homomorphism counts, we proceed in a similar way as Dvořák [11]: we use a variant of the *quantum graphs* of Lovász and Szegedy [20], tailored towards our setting. We say that  $k$ -labeled incidence graphs  $L_1, \dots, L_d$  are *compatible* if their labeling functions all have the same domain and they all have the same guard function, i.e.,  $\text{dom}(r_{L_1}) = \text{dom}(r_{L_i})$ ,  $\text{dom}(b_{L_1}) = \text{dom}(b_{L_i})$ , and  $g_{L_1} = g_{L_i}$  for all  $i \in [d]$ .

A  *$k$ -labeled quantum incidence graph*  $Q$  is a formal finite non-empty linear combination with real coefficients of compatible  $k$ -labeled incidence graphs. We represent a  $k$ -labeled quantum incidence graph  $Q$  as  $\sum_{i=1}^d \alpha_i L_i$ , where  $d \in \mathbb{N}_{\geq 1}$ ,  $\alpha_i \in \mathbb{R}$ , and  $L_i$  is a  $k$ -labeled incidence graph for  $i \in [d]$ . We let  $\text{dr}_Q := \text{dom}(r_{L_1}) = \dots = \text{dom}(r_{L_d})$ ,  $\text{db}_Q := \text{dom}(b_{L_1}) = \dots = \text{dom}(b_{L_d})$ , and  $g_Q := g_{L_1} = \dots = g_{L_d}$ . The  $\alpha_i$ 's and  $L_i$ 's are called the *coefficients* and *components*, respectively, and  $d$  is called the *degree* of  $Q$ . Note that a  $k$ -labeled incidence graph is a  $k$ -labeled quantum incidence graph with degree 1 and coefficient 1. For a  $k$ -labeled quantum incidence graph  $Q = \sum_{i=1}^d \alpha_i L_i$  and an arbitrary  $k$ -labeled incidence graph  $L'$  we let  $\text{hom}(Q, L') := \sum_{i=1}^d \alpha_i \cdot \text{hom}(L_i, L') \in \mathbb{R}$ .

We adapt the operations for  $k$ -labeled incidence graphs to their quantum equivalent in the expected way:  $Q[X_r \rightarrow \bullet] := \sum_{i=1}^d L_i[X_r \rightarrow \bullet]$ ,  $Q\langle X_b \rightarrow \bullet \rangle := \sum_{i=1}^d L_i\langle X_b \rightarrow \bullet \rangle$ ,  $Q[\rightsquigarrow f] := \sum_{i=1}^d \alpha_i L_i[\rightsquigarrow f]$ . Glueing two  $k$ -labeled quantum incidence graphs  $Q = \sum_{i=1}^d \alpha_i L_i$  and  $Q' = \sum_{j=1}^{d'} \alpha'_j L'_j$  is achieved by pairwise glueing of their components and multiplication of their respective coefficients, i.e.  $(Q \cdot Q') := \sum_{\substack{i \in [d] \\ j \in [d']}} (\alpha_i \cdot \alpha'_j) (L_i \cdot L'_j)$ .

The following can easily be proved for the case where  $Q, Q'$  have degree 1 and coefficient 1 (i.e.,  $Q, Q'$  are  $k$ -labeled incidence graphs), and then be generalised to quantum incidence graphs by simple linear arguments.

► **Lemma 7.5.** *For all  $k$ -labeled quantum incidence graphs  $Q, Q'$  and all  $k$ -labeled incidence graphs  $L$  we have:*

1.  $\text{hom}((Q \cdot Q'), L) = \text{hom}(Q, L) \cdot \text{hom}(Q', L)$ .
2.  $\text{hom}(Q[X_r \rightarrow \bullet], L) = \sum_{\bar{v} \in R(I_L)^\ell} \text{hom}(Q, L[X_r \rightarrow \bar{v}])$ , for all  $X_r \subseteq \text{dr}_Q$  and  $\ell := |X_r|$ .
3.  $\text{hom}(Q\langle X_b \rightarrow \bullet \rangle, L) = \sum_{\bar{e} \in B(I_L)^\ell} \text{hom}(Q, L\langle X_b \rightarrow \bar{e} \rangle)$ , for all  $X_b \subseteq \text{db}_Q$  and  $\ell := |X_b|$ .
4.  $\text{hom}(Q[\rightsquigarrow f], L) = \text{hom}(M_f, L) \cdot \sum_{\bar{e} \in B(I_L)^\ell} \text{hom}(Q, L\langle X_b \rightarrow \bar{e} \rangle)$ , for all transitions  $f$  for  $g_Q$ , for  $X_b := \text{db}_Q \cap \text{img}(f) \cap \text{img}(g)$  and  $\ell := |X_b|$ . Note that  $\text{hom}(M_f, L) \in \{0, 1\}$ .

The class  $QGLI_k$  of *guarded  $k$ -labeled quantum incidence graphs* consists of those  $k$ -labeled quantum incidence graphs where all components belong to  $GLI_k$ . The following lemma was provided for series-parallel quantum graphs by Lovász and Szegedy [20] and for labeled quantum graphs of tree-width  $\leq k$  by Dvořák [11]; their proof also works for  $QGLI_k$ .

► **Lemma 7.6.** *Let  $X, Y \subseteq \mathbb{N}$  be disjoint and finite, and let  $Q \in QGLI_k$ . There exists a  $Q[X, Y] \in QGLI_k$  with the same parameters  $\text{dr}_Q, \text{db}_Q, g_Q$  as  $Q$ , such that for all  $k$ -labeled incidence graphs  $L$  with real guards w.r.t.  $g_Q$  we have:*

1. If  $\text{hom}(Q, L) \in X$  then  $\text{hom}(Q[X, Y], L) = 0$ .
2. If  $\text{hom}(Q, L) \in Y$  then  $\text{hom}(Q[X, Y], L) = 1$ .

## 8 Proof of Theorem 6.1

Finally, we have available all the machinery so that, from a high-level point of view, our proof of Theorem 6.1 can follow a similar approach as Dvořák's proof in [11]. Analogously to the two main lemmas in [11], we provide a key lemma for each of the directions " $\Leftarrow$ " and " $\Rightarrow$ " of Theorem 6.1. These lemmas use the following notion: The *interpretation*  $\mathcal{I}_{L'}$  associated with a  $k$ -labeled incidence graph  $L'$  is an interpretation  $(I, \beta)$  with  $I := I_{L'}$  and  $\beta(\mathbf{v}_i) := r_{L'}(i)$  for all  $i \in \text{dom}(r_{L'})$  and  $\beta(\mathbf{e}_j) := b_{L'}(j)$  for all  $j \in \text{dom}(b_{L'})$ .

► **Lemma 8.1.** *Let  $L = (I, b, r, g) \in GLI_k$ . For every  $m \in \mathbb{N}$  there is a formula  $\varphi_{L,m}$  with  $(\Delta_g \wedge \varphi_{L,m}) \in \text{RGC}^k$  and  $\text{free}((\Delta_g \wedge \varphi_{L,m})) = \{\mathbf{v}_i : i \in \text{dom}(r)\} \cup \{\mathbf{e}_j : j \in \text{dom}(b)\}$  such that for every  $k$ -labeled incidence graph  $L' = (I', b', r', g')$  with  $\text{dom}(b_{L'}) \supseteq \text{dom}(b)$ ,  $\text{dom}(r_{L'}) \supseteq \text{dom}(r)$ , and with real guards w.r.t.  $g$  we have:  $\mathcal{I}_{L'} \models \Delta_g$ , and  $\text{hom}(L, L') = m \iff \mathcal{I}_{L'} \models \varphi_{L,m}$ .*

► **Lemma 8.2.** *Let  $\chi := (\Delta_g \wedge \psi) \in \text{RGC}^k$  and let  $m, d \in \mathbb{N}$  with  $m \geq 1$ . There exists a  $Q := Q_{\chi, m, d} \in QGLI_k$  with  $g_Q = g$ ,  $\text{db}_Q = \text{ifree}_B(\chi)$ ,  $\text{dr}_Q = \text{dom}(g) = \text{ifree}_R(\chi)$  such that for all  $k$ -labeled incidence graphs  $L' = (I', b', r', g')$  with  $|B(I')| = m$  and  $\max\{|N_{I'}(e)| : e \in B(I')\} \leq d$  and  $\text{dom}(b') \supseteq \text{db}_Q$ ,  $\text{dom}(r') \supseteq \text{dr}_Q$ ,  $g' \supseteq g$ , and with real guards w.r.t.  $g$  we have:  $\mathcal{I}_{L'} \models \Delta_g$ , and  $\text{hom}(Q, L') = 1$  if  $\mathcal{I}_{L'} \models \chi$ , and  $\text{hom}(Q, L') = 0$  if  $\mathcal{I}_{L'} \not\models \chi$ .*

The proofs of both lemmas are technically quite intricate because the concept of generalised hypertree width (as well as the classes  $IEHW_k$  and  $GLI_k$ ) is much more complicated than the concept of tree-width. For Lemma 8.1 we proceed by induction based on Definition 7.3; for Lemma 8.2 we proceed by induction on the construction of  $\chi$ . Finally, the proof of Theorem 6.1 can easily be achieved by using Theorem 7.4 and the Lemmas 8.1 (for direction " $\Leftarrow$ ") and 8.2 (for direction " $\Rightarrow$ ").

## 9 Conclusion

Combining the Theorems 3.1, 4.1, 6.1 yields:

► **Theorem 9.1** (Main Theorem). *Let  $H, H'$  be hypergraphs.*

(a)  $I_H \equiv_{\text{RGC}^k} I_{H'} \iff \text{HOM}_{\text{GHW}_k}(H) = \text{HOM}_{\text{GHW}_k}(H')$ .

(b) *If  $H$  and  $H'$  are simple, then:*  $I_H \equiv_{\text{RGC}^k} I_{H'} \iff \text{HOM}_{\text{sGHW}_k}(H) = \text{HOM}_{\text{sGHW}_k}(H')$ .

An obvious question is whether  $\text{RGC}^k$ -sentences have the same expressive power as  $\text{GC}^k$ -sentences. Since the submission of this paper, we were able to prove that this is indeed the case, i.e., any sentence of the logic  $\text{GC}^k$  can be transformed into an equivalent sentence in  $\text{RGC}^k$ . This implies that  $I_H \equiv_{\text{RGC}^k} I_{H'} \iff I_H \equiv_{\text{GC}^k} I_{H'}$ . Details are provided in the paper’s extended version [22].

For our proofs it was crucial to consider ehds instead of generalised hypertree decompositions. To the best of our knowledge, ehds have not been studied before. From Theorem 2.3 we know that there exist arbitrarily large  $k$  such that  $\text{IEHW}_k$  is a strict subclass of  $\text{IGHW}_k$ ; but nevertheless, according to Theorem 4.1 homomorphism indistinguishability coincides for both classes. Many other questions remain open, in particular: How hard is it, given a hypergraph  $H$  and a number  $k$ , to determine whether  $\text{ehw}(H) \leq k$ ? For  $\mathcal{C} := \text{IEHW}_k$ : how hard is it to compute the function (or, “vector”)  $\text{HOM}_{\mathcal{C}}(H)$  for a given hypergraph  $H$ ? Which properties does it have? What is the expressive power of the logic  $\text{GC}^k$ ? How does a suitable pebble game for  $\text{GC}^k$  look like? Our result lifts Dvořák’s result for tree-width  $\leq k$  [11] from graphs to hypergraphs. Does there also exist a lifting of Grohe’s result for *tree-depth*  $\leq k$  [16] from graphs to hypergraphs? Seeing that Dvořák’s result lifted nicely to hypergraphs, we believe that there should also be a lifting of Cai, Fürer and Immerman’s result [8], i.e., a hypergraph-variant of the Weisfeiler-Leman algorithm, whose distinguishing power matches precisely the logic  $\text{GC}^k$ . We plan to study this in future work.

---

## References

- 1 Isolde Adler. Marshals, monotone marshals, and hypertree-width. *J. Graph Theory*, 47(4):275–296, 2004. doi:10.1002/jgt.20025.
- 2 Isolde Adler. *Width Functions for Hypertree Decompositions*. PhD thesis, Albert-Ludwigs Universität Freiburg, 2006. Available at <https://d-nb.info/979896851/34>.
- 3 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007. doi:10.1016/j.ejc.2007.04.013.
- 4 Hajnal Andr eka, Istv an N emeti, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Log.*, 27(3):217–274, 1998. doi:10.1023/A:1004275029985.
- 5 Jan B oker, Yijia Chen, Martin Grohe, and Gaurav Rattan. The complexity of homomorphism indistinguishability. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26–30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 54:1–54:13. Schloss Dagstuhl – Leibniz-Zentrum f ur Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.54.
- 6 Jan B oker. Structural similarity and homomorphism counts. Master’s thesis, RWTH Aachen University, 2018.
- 7 Jan B oker. Color Refinement, Homomorphisms, and Hypergraphs. In Ignas Sau and Dimitrios M. Thilikos, editors, *Graph-Theoretic Concepts in Computer Science*, volume 11789 of *Lecture Notes in Computer Science*, pages 338–350. Springer, Cham, 2019.
- 8 Jin-Yi Cai, Martin F urer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, December 1992. doi:10.1007/BF01305232.

- 9 Bruno Courcelle. Graph Grammars, Monadic Second-Order Logic And The Theory Of Graph Minors. In Neil Robertson and Paul Seymour, editors, *Graph Structure Theory*, volume 147 of *Contemporary Mathematics*, pages 565–590. AMS, 1993.
- 10 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász meets Weisfeiler and Leman. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 40:1–40:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.40.
- 11 Zdeněk Dvořák. On Recognizing Graphs by Numbers of Homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010.
- 12 Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: Questions and answers. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 – July 01, 2016*, pages 57–74. ACM, 2016.
- 13 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.
- 14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, June 2003.
- 15 Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999. doi:10.2307/2586808.
- 16 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS’20*, pages 507–520. ACM, July 2020. doi:10.1145/3373718.3394739.
- 17 Martin Grohe. Word2vec, Node2vec, Graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS’20*, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.
- 18 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, October 2000.
- 19 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungaricae*, 18(3):321–328, 1967.
- 20 László Lovász and Balázs Szegedy. Contractors and Connectors of Graph Algebras. *Journal of Graph Theory*, 60(1):11–30, 2008.
- 21 Laura Mancinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 661–672. IEEE, 2020. doi:10.1109/FOCS46700.2020.00067.
- 22 Benjamin Scheidt and Nicole Schweikardt. Counting homomorphisms from hypergraphs of bounded generalised hypertree width: A logical characterisation. *CoRR*, abs/2303.10980, 2023. doi:10.48550/arXiv.2303.10980.



# Dynamic Constant Time Parallel Graph Algorithms with Sub-Linear Work

Jonas Schmidt ✉

TU Dortmund University, Germany

Thomas Schwentick ✉

TU Dortmund University, Germany

---

## Abstract

---

The paper proposes dynamic parallel algorithms for connectivity and bipartiteness of undirected graphs that require constant time and  $\mathcal{O}(n^{1/2+\epsilon})$  work on the CRCW PRAM model. The work of these algorithms almost matches the work of the  $\mathcal{O}(\log n)$  time algorithm for connectivity by Kopelowitz et al. (2018) on the EREW PRAM model and the time of the sequential algorithm for bipartiteness by Eppstein et al. (1997). In particular, we show that the sparsification technique, which has been used in both mentioned papers, can in principle also be used for constant time algorithms in the CRCW PRAM model, despite the logarithmic depth of sparsification trees.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Dynamic graph algorithms; Theory of computation  $\rightarrow$  Parallel algorithms

**Keywords and phrases** Dynamic parallel algorithms, Undirected connectivity, Bipartiteness

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.80

**Related Version** *Full Version:* <http://arxiv.org/abs/2307.10107>

**Acknowledgements** We are grateful to Jens Keppeler and Christopher Spinrath for careful proof reading.

## 1 Introduction

There has been a lot of research on dynamic algorithms for graph problems.<sup>1</sup> Usually, the setting is that graphs can be changed by edge insertions or deletions and that there are query operations that allow to check whether the graph has certain properties. Most of this research has been about sequential algorithms and the goal has been to find algorithms that are as fast as possible. Some algorithms use randomisation, others are deterministic, sometimes the time bounds are worst-case bounds per change or query operation and sometimes they are amortised bounds.

There has been also some research on dynamic *parallel* graph algorithms. Many of these algorithms use the EREW PRAM model<sup>2</sup> and try to achieve logarithmic or polylogarithmic running time, while being work-efficient or even work-optimal. That is, the overall work of all processors should be (almost) the same as for the best sequential algorithm.<sup>3</sup>

There is an entirely separate line of work that studied the maintenance of graph (and other) properties in a setting that was inspired by Database Theory. It is often called *Dynamic Complexity* in Database Theory. In the setting of Dynamic Complexity, dynamic algorithms are called *dynamic programs* and they are not specified in an “algorithmic fashion”

---

<sup>1</sup> Below we will give pointers to literature. For the beginning of the introduction, we try to keep the story simple.

<sup>2</sup> In an EREW PRAM, parallel processors can use shared memory, but at each moment, each memory cell can be accessed by only one processor. EREW stands for *exclusive-read/exclusive-write*.

<sup>3</sup> We note that in our context of constant-time parallel algorithms work is within a constant factor of the number of processors.



© Jonas Schmidt and Thomas Schwentick;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 80; pp. 80:1–80:15

Leibniz International Proceedings in Informatics



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

but rather by logical formulas. As a classical example from [13], to maintain reachability information between pairs of nodes in a directed acyclic graph, a dynamic program can use an auxiliary relation  $T$  that is intended to store the transitive closure of the graph. The program can then be specified by two formulas that specify how the new version  $T'$  of  $T$  is defined after the insertion or deletion of an edge  $(u, v)$ :

**Insertion:**  $T'(x, y) \stackrel{\text{def}}{=} T(x, y) \vee (T(x, u) \wedge T(v, y))$ . After inserting  $(u, v)$  there is a path from  $x$  to  $y$  if such a path already existed or if there was a path from  $x$  to  $u$  and from  $v$  to  $y$ .

**Deletion:**  $T'(x, y) \stackrel{\text{def}}{=} T(x, y) \wedge \left( E(x, y) \vee \neg T(x, u) \vee \neg T(v, y) \vee \exists u', v' \left( (u' \neq u \vee v' \neq v) \wedge T(x, u') \wedge E(u', v') \wedge T(v', y) \wedge T(u', u) \wedge \neg T(v', u) \right) \right)$ .

This formula is slightly more complicated. In the main case, the nodes  $u', v'$  are chosen such that neither the path from  $x$  to  $u'$  nor the path from  $v'$  to  $y$  relies on the edge  $(u, v)$ . In the former case this is thanks to  $T(u', u)$  (since if  $T(x, u')$  involved  $(u, v)$ , the graph were not be acyclic) and in the latter case it is thanks to  $\neg T(v', u)$ .

As in the example, the underlying logic is usually first-order logic, since it corresponds to the main (theoretical) query language for relational databases, the relational algebra, which in turn corresponds to the core of SQL. The class of problems or queries that can be maintained in this way is usually called DynFO.

Dynamic Complexity has existed quite separated from the world of dynamic algorithms, but there is a direct link that connects the two areas: it follows from a fundamental result<sup>4</sup> from Immerman [8, Theorem 1.1] that dynamic programs can be translated into parallel programs that run in *constant time* on suitable versions of CRCW PRAMs<sup>5</sup> with polynomially many processors. And vice versa.

Dynamic Complexity has focussed on the question whether a graph property can be maintained at all by first-order logic (or fragments thereof), but did not care about the work efficiency of the parallel algorithms that are obtained from translating the update formulas. It turns out that this automatic translation often does not yield very efficient parallel algorithms.

As an example, the parallel dynamic algorithm that is obtained by direct translation of the above formulas, has work  $\mathcal{O}(n^4)$  for deletions, since it would consist of two nested loops for  $x$  and  $y$  and two more for  $u$  and  $v$ . This is far from being work-efficient.<sup>6</sup> The translation of the dynamic program for Connectivity in undirected graphs from [13] even yields a work bound of  $\mathcal{O}(n^5)$ . We will show that this work bound can be improved considerably.

This paper is part of an effort to bridge the gap between Dynamic Complexity and (parallel) Dynamic Algorithms by developing algorithms that run in constant time on CRCW PRAMs and are as work efficient as possible. It presents constant-time dynamic parallel algorithms for Connectivity and Bipartiteness in undirected graphs. In the *arbitrary* CRCW model, the algorithms require work at most  $n^{\frac{1}{2}} \text{polylog}(n)$ . In the *common* CRCW model, the algorithms can be instantiated, for each constant  $\epsilon > 0$ , such that they obey a work bound of  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$ , where  $n$  is the number of nodes in the graph.

<sup>4</sup> Immerman's result is not about dynamic programs, but each formula of a dynamic program can be translated separately.

<sup>5</sup> In a CRCW PRAM more than one processor can read a memory cell, at the same time. Even more than one processor can write into the same cell, but there has to be a strategy that deals with conflicts. This will be explained later in the text.

<sup>6</sup> We have to admit that this paper does not present a better algorithm for directed reachability. We chose that problem only as an example, since its formulas are relatively easy to understand.

The algorithm for Connectivity follows the parallel EREW PRAM algorithm of Kopelowitz et al. [12], which in turn was based on a sequential algorithm by Fredrickson [5] and its sparsification by Eppstein et al. [4]. Thus, the work of our algorithm almost matches the work bound  $\mathcal{O}(n^{\frac{1}{2}})$  of the parallel algorithm of [12] and the worst-case runtime of [4]. However, it does not match the runtime of the recent breakthrough algorithm by Chuzhoy et al. [1].

The main technical challenge here is to make the sparsification and the tree-like data structure of [12] work in constant time, despite their use of trees of logarithmic depth. For sparsification, this means updating all logarithmically many nodes along the path from the changed leaf to the root of a tree of logarithmic height in parallel constant time although in classical sparsification the change in the leaf is propagated from one node to the other along the path. For handling the tree-like data structure of [12] in constant parallel time, data is stored differently by switching from lists to arrays and it is shown (in full version of this paper) that balanced search trees ( $(a, b)$ -trees to be precise) of logarithmic height are maintainable in constant parallel time. In the classical algorithm for, e.g., splitting an  $(a, b)$ -tree into two separate trees, the tree is first split into logarithmically many smaller trees and then the two new trees are built by merging logarithmically many of those smaller trees back together. Both steps are done sequentially in  $\mathcal{O}(\log n)$  time by splitting one of those smaller trees at a time and merging only two of the smaller trees at a time, but for our purpose have to be done in constant parallel time.

The algorithm for bipartiteness almost matches the runtime of the bipartiteness algorithm of Eppstein et al. [4]. It is based on the observation that a graph is bipartite if and only if its distance-2 graph has twice as many connected components as the graph itself. The algorithm therefore basically maintains two spanning trees, for the graph and its distance-2 graph. Here, the main technical challenge is to show that the same sparsification approach as for connectivity also works for bipartiteness.

**Structure of the paper.** We introduce some basic concepts about CRCW PRAMs in Section 2. The algorithm for connectivity is presented in Section 3. The algorithm for bipartiteness is given in Section 4.

**Related work.** Some related work has already been mentioned above. Dynamic Complexity has started by the work of Patnaik and Immerman [13] and Dong and Su [3]. For a recent survey on the dynamic complexity of Reachability in directed and undirected graphs, we refer to [14]. For a recent survey on dynamic graph algorithms, we refer to [6].

Of course, the PRAM model is not the only parallel computation model for parallel algorithms. Parallel dynamic algorithms for the MPC model can be found, e.g., in [9].

## 2 Preliminaries

For natural numbers  $i \leq j$ , we write  $[i, j]$  for the set  $\{i, \dots, j\}$ . We only deal with undirected graphs and denote an undirected edge between two vertices  $u$  and  $v$  by  $(u, v)$ .

**Dynamic algorithmic problems.** In this paper, we view a dynamic (algorithmic) problem basically as the interface of a data type: that is, there is a collection of operations by which some object can be initialised, changed, and queried. A *dynamic algorithm* is then a collection of algorithms, one for each operation. We consider two main dynamic problems in this paper, CONNECTIVITY and BIPARTITENESS.

The algorithmic problem CONNECTIVITY maintains an undirected graph  $G$  and has the following operations.

- `Init( $G, n$ )` yields an initial graph  $G$  with  $n$  nodes, that are initially deactivated but without edges;
- `ActivateNode( $G, v$ )` yields an identifier for a new node of  $G$ ;
- `DeactivateNode( $G, v$ )` deactivates the node  $v$  from  $G$ . The node  $u$  must be isolated;
- `InsertEdge( $G, u, v$ )` inserts edge  $(u, v)$  to  $G$ ;
- `DeleteEdge( $G, u, v$ )` deletes edge  $(u, v)$  from  $G$ ;
- `Connected( $G, u, v$ )` returns true if  $u$  and  $v$  are in the same connected component, otherwise false.
- `#Components( $G$ )` yields the number of connected components of  $G$  on the activated nodes.

BIPARTITENESS has almost the same operations, but instead of `Connected` and `#Components` it has a query operation `Bipartite( $G$ )` which yields true if the graph  $G$  is bipartite.

Throughout this paper we only consider the effort for change and query operations, but disregard the effort for the initialisation of a graph. We also note that the number  $n$  of nodes can not grow. The nodes are represented by numbers in  $\{1, \dots, n\}$ .

**Parallel Random Access Machines (PRAMs).** A *parallel random access machine* (PRAM) consists of a number of processors that work in parallel and use a shared memory.<sup>7</sup> The memory is comprised of memory cells which can be accessed by a processor in  $\mathcal{O}(1)$  time. Furthermore, we assume that simple arithmetic and bitwise operations, including addition, can be done in  $\mathcal{O}(1)$  time by a processor. The work of a PRAM computation is the sum of the number of all computation steps of all processors made during the computation. We define the space  $s$  required by a PRAM computation as the maximal index of any memory cell accessed during the computation.

We use the Concurrent-Read Concurrent-Write model (CRCW PRAM), i.e., processors are allowed to read and write concurrently from and to the same memory location. More precisely, we will consider two different versions of CRCW PRAMs.

- In the *arbitrary* model, if multiple processors concurrently write to the same memory location, one of them, “arbitrarily”, succeeds;
- In the slightly weaker *common* model, concurrent write into the same memory location, is only allowed if all processors write the same value.

The two models will yield slightly different work bounds for our dynamic algorithms for CONNECTIVITY and BIPARTITENESS: in the arbitrary model, the work will be at most  $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$ , whereas in the common model, we will have algorithms with work  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$ , for every  $\epsilon > 0$ . Here,  $\tilde{\mathcal{O}}(f(n))$  allows an additional polylogarithmic factor with  $f(n)$ .

We refer to [10] for more details on PRAMs and to [15, Section 2.2.3] for a discussion of alternative space measures.

For simplicity, we assume that even if the number  $n$  of nodes of the input graph grows, a number in the range  $[0, n]$  can still be stored in one memory cell. This assumption is justified, since addition of larger numbers  $N$  can still be done in constant time and polylogarithmic work on a CRCW PRAM.

The following lemma exhibits a simple CRCW PRAM algorithm in the common model that will be used as a sub-algorithm. It also illustrates the frequent use of arrays in PRAM algorithms. It will mainly be used as a tie-breaker, if one of several objects has to be chosen, and it will therefore not be needed in the context of the arbitrary model. The lemma was shown in a slightly more general form in [11, Proposition 5.4].

---

<sup>7</sup> Some content of this paragraph is copied from [11].

► **Lemma 2.1** ([11, Proposition 5.4]). *Let  $A$  be an array of size  $n$  over a finite alphabet  $\Sigma$ . The minimum/maximum value of  $A$  can be computed in constant parallel time on a common CRCW PRAM with work  $\mathcal{O}(n^{1+\epsilon})$  for any  $\epsilon > 0$ .*

**Proof sketch.** We only describe how the minimum can be computed, since finding the maximum value is completely analogous. A naïve approach is to assign one processor to each pair  $i, j$  of positions in the array. Whenever  $A[i] < A[j]$  or  $A[i] = A[j]$  and  $i < j$ , then a 1 is written into  $B[j]$ , where  $B$  is an auxiliary array of size  $n$ , in which all entries are initially set to zero. Afterwards, one processor is assigned to each cell of  $B$  and the processor assigned to the only cell  $B[i]$  with value 0 outputs  $A[i]$  as the minimum. However, this algorithm requires  $\mathcal{O}(n^2)$  work. The (standard) idea to reduce the work to  $\mathcal{O}(n^{1+\epsilon})$  is to first compute the minimum of subarrays of  $A$  of size  $n^\epsilon$ . This requires time  $\mathcal{O}(n^{2\epsilon})$ , for each of the  $n^{1-\epsilon}$  subarrays, resulting in work  $\mathcal{O}(n^{1+\epsilon})$ . The minimal values of the sub-arrays can then be stored in an array of size  $n^{1-\epsilon}$  whose minimum can be computed recursively. Since the number of recursion rounds is bounded by the constant  $\lceil \frac{1}{\epsilon} \rceil$ , the overall work is  $\mathcal{O}(n^{1+\epsilon})$ . ◀

### 3 Connectivity

In this section, we present the main result of this paper and (most of) its proof.

► **Theorem 3.1.** *There are dynamic parallel constant-time algorithms for CONNECTIVITY with the following work bounds per change or query operation.*

- $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$  work on the arbitrary CRCW PRAM model.
- $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$  work on the common CRCW PRAM model, for every  $\epsilon > 0$ .

As usual, the algorithm basically maintains a spanning forest and the graph  $G$  is connected if and only if  $\#\text{Components}(G)$  yields 1.

In fact, we will consider the data type SPANNINGFOREST as an extension of CONNECTIVITY with the following additional operation.

- $\text{TreeEdge}(G, u, v)$  returns true if  $(u, v)$  is a tree edge, otherwise false.

The proof is along the lines of [12] and is split into the same three main steps. For each step, we need to show that it can be done in constant parallel time on a CRCW PRAM, as opposed to  $\mathcal{O}(\log n)$  on an EREW PRAM. This strengthening comes with an additional work factor of  $\text{polylog}(m)$  or  $\text{polylog}(n)$  on an arbitrary CRCW PRAM and  $m^\epsilon$  or  $n^\epsilon$  on a common CRCW PRAM.

We first show that, for graphs of maximum degree 3, SPANNINGFOREST can be maintained with work  $\tilde{\mathcal{O}}(m^{\frac{1}{2}})$  and  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  per operation, depending on the PRAM model. Then we show that the case of graphs of unbounded degree can be reduced to the case of graphs with degree bound three. Finally, we show that, with the help of sparsification, both bounds from above are translatable to be in  $n$  instead of  $m$ .

More precisely, we show the following three results.

► **Proposition 3.2.** *There are dynamic parallel constant time algorithms for the special case of SPANNINGFOREST, where the maximum degree of the graph never exceeds 3 with the following work bounds per change or query operation.*

- $\tilde{\mathcal{O}}(m^{\frac{1}{2}})$  on the arbitrary CRCW PRAM model.
- $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  on the common CRCW PRAM model, for every  $\epsilon > 0$ .

► **Proposition 3.3.** *If SPANNINGFOREST can be maintained in parallel constant time on a CRCW PRAM with the work bounds of Proposition 3.2 per change or query operation, for any  $\epsilon > 0$ , for graphs with maximum degree 3, it can be maintained with the same bounds for general graphs with the provision that they never have more than  $cn$  edges, for some constant  $c$ .*

► **Proposition 3.4.** *If SPANNINGFOREST can be maintained in parallel constant time on a CRCW PRAM with the work bounds of Proposition 3.2 per change or query operation, then it can also be maintained with  $\tilde{O}(n^{\frac{1}{2}})$  work per change or query operation on the common CRCW model and with  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$  work per change or query operation on the arbitrary CRCW model.*

Proposition 3.2 will be shown in the next two subsections. Proposition 3.3 and Proposition 3.4 will be shown in Subsection 3.2.

### 3.1 Maintaining a spanning forest for bounded degree graphs

As mentioned before, our algorithm closely follows [12] and therefore uses a similar data structure. Some modifications are required though, to achieve constant parallel update and query time while keeping almost the same amount of work. The data structure maintains an Euler tour, for each spanning tree in a spanning forest of the graph. More precisely, it maintains, for each spanning tree, a cyclic list of tree edges that visits each tree edge once in either direction.

We first concentrate on the change operations `InsertEdge`( $G, u, v$ ) and `DeleteEdge`( $G, u, v$ ) and the query operations.

The algorithm does not need to change the Euler tour, if a new edge is inserted which connects two nodes of the same spanning tree or if a non-tree edge is deleted. If an edge  $e$  between two different spanning trees is inserted, the algorithm can just merge the two Euler tours. If an edge  $e$  of a spanning tree is deleted, the algorithm first splits the Euler tour at both occurrences of  $e$  and then tries to find a *replacement edge* that connects the two sub-trees resulting from the deletion. The search for a replacement edge is actually the most critical part of the algorithm, since trying out all edges of the graph would yield linear work.

Towards a more efficient algorithm, we follow the same two-tiered approach as [12]: each Euler tour is chopped into chunks of about  $\sqrt{m}$  edges, which are represented as *arrays of edges*. The underlying idea is that after a change operation the necessary updates can be divided into low-level manipulations inside only a few chunks and high-level manipulations on the level of sequences of chunks. Each kind of manipulation should cause not much more than  $\mathcal{O}(\sqrt{m})$  work.

Furthermore, it will maintain information about non-tree edges between different chunks, ultimately allowing to find a replacement edge with work close to  $\mathcal{O}(\sqrt{m})$ .

We fix a number  $K$  that will be roughly  $\sqrt{m}$  later on and enforce that chunks contain between  $\frac{K}{2}$  and  $K$  edges, with the exception of at most one chunk per spanning tree. We denote the number of chunks by  $J$  which is in  $\mathcal{O}(\frac{m}{K})$ .

For the lower tier, i.e., creating and removing chunks and changing their content and additional information, the edge arrays representing the chunks are stored together in one *master array*  $\mathcal{M}$  with  $\mathcal{O}(\sqrt{m})$  slots of sub-arrays of length  $K$ . The slots will contain some additional information to be specified later. The order of chunks in  $\mathcal{M}$  can be arbitrary and  $\mathcal{M}$  might contain empty slots from deleted chunks. By  $\mathcal{M}(i)$  we refer to the chunk that is stored in the  $i$ -th slot of the master array. Some entries in  $\mathcal{M}$  might be unused or deactivated. For a chunk  $C$ , we refer by  $C$  also to the entry in  $\mathcal{M}$  for this chunk.

We say that two chunks  $C$  and  $C'$  are *linked*, if there is a non-tree edge  $(u, v)$  in  $G$  such that  $u$  occurs in  $C$  and  $v$  in  $C'$ . With each chunk  $C$  of edges, we associate a *link vector*  $B_C$ , which is a bit array of length  $J$  that reflects which chunks are linked with  $C$ . More precisely,  $B_C(i) = 1$  if  $C$  and  $\mathcal{M}(i)$  are linked. Here all slots in  $\mathcal{M}$  are relevant, even the unused or deactivated ones (but they will inevitably yield the bit 0).

The higher tier, which is responsible for maintaining the order of the chunks and information about sequences of chunks, maintains, for each Euler tour a *chunk array*: this is an array of pointers to chunks such that the concatenation of all edge lists in the order induced by the array represents the Euler tour. Furthermore, it maintains information about links between sequences of chunks in a sufficiently work-efficient way.

The algorithm uses a data type `CHUNKARRAYS` whose operations can be split into two groups. The first group consists of the following operations, which only access chunks and their link arrays, but do not directly refer to chunk arrays. In both groups of operations,  $i, j$  and  $k$  are always indices,  $C, C_1$  and  $C_2$  are chunk (pointers),  $A, A_1$  and  $A_2$  are (pointers to) chunk arrays,  $B$  is a bit vector and  $E$  is an edge array.

- `SetChunk( $i, C, E$ )` activates a new chunk  $C$  in  $\mathcal{M}(i)$  and stores the edge array  $E$  in  $C$ ;
- `Deactivate( $C$ )` deactivates chunk  $C$  in  $\mathcal{M}$ ;
- `Link( $C_1, C_2$ )` and `Unlink( $C_1, C_2$ )` allow to mark chunks  $C_1$  and  $C_2$  as linked or unlinked;
- `BulkSetLinks( $C, B$ )` replaces the link vector of chunk  $C$  by the bit vector  $B$  and changes the bit that refers to  $C$  in all other chunks  $C'$  according to  $B$ . More precisely, if  $C = \mathcal{M}(i)$  then, for each  $j$ , the  $i$ -th bit in the link vector of  $\mathcal{M}(j)$  is set to  $B(j)$ .

We note that `SetChunk` and `Deactivate` do not automatically change any link vectors.

The operations of the other group are as follows. They explicitly refer to chunk arrays.

- `InsertChunk( $A, i, C$ )` inserts (a pointer to) chunk  $C$  at position  $i$  of chunk array  $A$ , moving each entry, from  $i$  on, by one to the right;
- `DeleteChunk( $A, i$ )` deletes the chunk pointer at position  $i$  of chunk array  $A$ , moving each entry, from  $i + 1$  on, one to the left;
- `Concatenate( $A_1, A_2$ )` concatenates the chunk array  $A_2$  to the end of  $A_1$ ;
- `Split( $A, i$ )` splits  $A$  into two arrays  $A_1$  and  $A_2$  getting intervals  $[1, i]$  and  $[i + 1, \max(A)]$  and yields (pointers to)  $A_1$  and  $A_2$ ;
- `Reorder( $A, i, j, k$ )` moves the chunks of positions  $j, \dots, k$  to position  $i < j$ . That is, the chunks are ordered as  $1, \dots, i - 1, j, \dots, k, i + 1, \dots, j - 1, k + 1, \dots, m$ , where  $m$  is the size of  $A$ ;
- `Query( $A, i, j, k, \ell$ )` yields an arbitrary pair  $(C, C')$  of linked chunks where  $C$  is from  $[i, j]$  and  $C'$  is from  $[k, \ell]$ .

The algorithm will maintain the invariant that each chunk that is present in  $\mathcal{M}$  occurs in at most one chunk array. Each chunk in the master array contains a back pointer to its chunk pointer in its chunk array, and these entries are maintained by the above operations.

The following lemma is shown in the full version of this paper.

► **Lemma 3.5.** *There is a dynamic parallel constant-time algorithm for `CHUNKARRAYS` on an arbitrary `CRCW PRAM` that supports all operations with  $\mathcal{O}(J \text{ polylog } J)$  work.*

*Furthermore, for each  $\epsilon > 0$ , there is a parallel constant-time dynamic algorithm for `CHUNKARRAYS` on a common `CRCW PRAM` that supports `Query` with  $\mathcal{O}(J^{1+\epsilon})$  work and all other operations with  $\mathcal{O}(J \text{ polylog } J)$  work.*

The implementation of `CHUNKARRAYS` uses  $(a, b)$ -trees [7], which are trees of logarithmic height in which inner nodes have between  $a$  and  $b$  children, and support insertion and deletion of leaves as well as split and join of trees. In the `CHUNKARRAYS` for each chunk array  $A$  one  $(2, 6)$ -tree is maintained, that has the link arrays of the chunks of  $A$  at its leaves, in the order of  $A$ . The inner vertices of the tree store link arrays that are the bitwise disjunction of the link arrays of the leaves below them.

Given Lemma 3.5, we can now show Proposition 3.2, stating that SPANNINGFOREST can be maintained in parallel constant time with  $\tilde{O}(m^{\frac{1}{2}})$  work per operation on an *arbitrary* CRCW PRAM and, for every  $\epsilon > 0$ , with work  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  work per operation, on a *common* CRCW PRAM, if the maximum degree of the graph never exceeds 3. The degree bound mainly helps by bounding the number of edges incident to one chunk by  $\mathcal{O}(K)$ .

**Proof (of Proposition 3.2).** We first describe the data structure and then how it is maintained for the different change operations.

The algorithm maintains a master array and a chunk array as described above. It uses them to maintain a spanning tree and a corresponding Euler tour for each connected component of the graph. It uses  $K = J = \sqrt{m}$ .

Furthermore, the algorithm maintains an array with all nodes of the graph, and three additional entries for the up to three neighbours of each node, representing the edges. Additionally, there are pointers to the at most six appearances of a node in edges of Euler tours in the master array. The algorithm thereby implicitly maintains pointers from each edge to its at most two appearances in the Euler tours. Finally, a counter for the number of connected components is maintained.

The two query operations `Connected` and `#Components` can be answered in constant sequential time using the pointers from each node to occurrences in an Euler tour or the maintained counter, respectively.

For the change operation `InsertEdge`( $u, v$ ), we consider two different cases: (1) the insertion of a new edge ( $u, v$ ) where  $u$  and  $v$  are in the same connected component and (2) the insertion of a new edge ( $u, v$ ) where  $u$  and  $v$  are in different connected components.

The algorithm first identifies through the master array two chunk arrays  $A_u$  and  $A_v$  in which  $u$  and  $v$  reside. If  $A_u = A_v$ , we are in case (1) and it suffices to mark  $C_u$  and  $C_v$  as linked by `Link`( $C_u, C_v$ ) for all  $C_u$  and  $C_v$  that contain  $u$  and  $v$  respectively. All those chunks can be found using the maintained pointers from nodes to their appearances in chunks.

If  $A_u$  and  $A_v$  are different, we are in case (2) and ( $u, v$ ) newly connects the two spanning trees  $T_u$  and  $T_v$ , yielding a new spanning tree  $T$ . To this end, the two Euler tours represented by  $A_u$  and  $A_v$  need to be joined. Let the tour of  $A_u$  consist of two paths  $P_1, P_2$ , where  $P_1$  ends in  $u$  and  $P_2$  starts in  $u$ . Note that the last node of  $P_2$  is the same as the first node of  $P_1$ . Let the two paths  $Q_1, Q_2$  be defined analogously for  $A_v$  and  $v$ . Then the combined Euler tour will be  $P_1, (u, v), Q_2, Q_1, (v, u), P_2$ .

The algorithm first joins the two chunk arrays by `Concatenate`( $A_u, A_v$ ). It splits the edge sequence of  $C_u$  into a sequence  $E_u^1$  that ends in  $u$  and the remaining sequence  $E_u^2$  that starts in  $u$ . The position where  $C_u$  needs to be split can be found using the maintained edge pointers.  $E_u^1$  remains in  $C_u$  and for  $E_u^2$  a new chunk  $C'_u$  is reserved in  $\mathcal{M}$  and inserted in  $A_u$ , next to  $C_u$ . Similarly, the content of  $C_v$  is split into  $C_v$  and a new chunk  $C'_v$ . Then ( $u, v$ ) is added to  $C_u$  and ( $v, u$ ) to  $C_v$ .

The algorithm then restructures  $A_u$  by copying the sub-arrays corresponding to  $Q_2$  and ( $Q_1, (v, u)$ ) to their correct places by two calls to `Reorder`. This also moves  $P_2$  to its right place. If any of the four modified chunks has fewer than  $\frac{K}{2}$  edges, it is combined with a neighbour chunk in  $A_u$ : if possible, the two chunks are joined or otherwise each gets at least  $\frac{K}{2}$  edges to fulfil the invariant. This completes the restructuring of  $A_u$ .

It remains to update the link information between chunks. To this end, the algorithm first computes the link vectors for  $C_u, C'_u, C_v$  and  $C'_v$ . This can be done by initializing the vector with  $\vec{0}$  and then scanning all at most  $3K$  edges of the respective chunk. The four resulting link vectors are then set by `BulkSetLinks`. The operation `BulkSetLinks` takes also care of the modifications in the link vectors of all other chunks. Finally, the connected component counter is decreased by 1.

For the change operation `DeleteEdge`( $u, v$ ), we consider again two different cases: (1) the deletion of an edge  $(u, v)$  that is not in any spanning tree and (2) the deletion of a spanning tree edge  $(u, v)$ . If there is no pointer from  $(u, v)$  to an occurrence in any Euler tour, we are in case (1) and  $(u, v)$  was not a spanning tree edge. The algorithm then checks whether the chunk pairs linked by  $(u, v)$  are still linked without  $(u, v)$  by scanning all, thanks to the degree bound at most  $3K$  many edges of one of the two chunks per pair. If not, it marks the two chunks as unlinked with `Unlink`.

If there are pointers from  $(u, v)$  to occurrences in an Euler tour, we are in case (2) and  $(u, v)$  was a spanning tree edge. Let  $P_1, (u, v), P_2, (v, u), P_3$  be the decomposition of the Euler tour of its tree  $T$ . By two applications of `Query` the algorithm checks whether there are any links between  $P_1$  and  $P_2$  or  $P_3$  and  $P_2$ , respectively.

If there are no such edges then it first reorders the chunk array, so that  $P_1$  and  $P_3$  are consecutive, moving  $P_2$  towards the end, and then splits it into the two parts  $P_1, P_3$  and  $P_2$ .

If there is such an edge, let us assume there are chunks  $C_1$  in  $P_1$  and  $C_2$  in  $P_2$  that are linked. The algorithm inspects all edges in  $C_1$  in parallel and tests, whether their partner edge is in  $C_2$ . It then either picks an arbitrary edge (if the PRAM model supports that) or computes the minimum edge with this property. Let the chosen replacement edge be  $(w_1, w_2)$ . Decomposing  $P_1$  into  $P'_1, P''_1$ , separated at  $w_1$ , and  $P_2$  into  $P'_2, P''_2$ , separated at  $w_2$ , the new Euler cycle is  $P'_1, (w_1, w_2), P'_2, P''_2, (w_2, w_1), P''_1, P_3$ . It can be constructed in  $T$  by splitting  $C_1$  and  $C_2$  into two chunks, with the help of two newly inserted chunks, reordering the array, and repairing small chunks, similarly to the above case of inserting a new tree edge.

The work of the algorithm is dominated by finding a replacement edge. It requires two initial calls to `Query` of the `CHUNKARRAYS` data type requiring  $\mathcal{O}(J \text{ polylog } J)$  or  $\mathcal{O}(J^{1+\epsilon})$  work, depending on the PRAM model. Then it requires work  $\mathcal{O}(K)$  to identify all at most  $3K$  possible replacement edges. In the arbitrary model the choice of the actual edge is immediate. In the common model, it might take work  $\mathcal{O}(K^{1+\epsilon})$ . Apart from that, the algorithm applies a constant number of calls to operations of `CHUNKARRAYS` that all require  $\mathcal{O}(J \text{ polylog } J)$  work. By Lemma 3.5 and the choice of  $J$  and  $K$  we get the desired work bounds.

The operations `ActivateNode` and `DeactivateNode` can be easily implemented in constant sequential time. ◀

### 3.2 From bounded to unbounded degree and from $m$ to $n$

We first show Proposition 3.3 which allows us to conclude that Proposition 3.2 can be lifted to graphs without a degree restriction.

**Proof sketch (of Proposition 3.3).** Like for [12] our algorithm uses the well known graph reduction already used by [5]. To maintain connectivity for an unrestricted graph  $G$ , the algorithm maintains a graph  $G'$  of degree at most 3, which is connected if and only if  $G$  is connected. The number of nodes of this graph is initialised as  $cn$ , where  $c$  is as in the statement of the proposition. The idea of the reduction is to replace each node  $v$  of  $G$  of degree  $d > 3$  in  $G'$  by a cycle of length  $d$  and to connect each node of the cycle to one node adjacent to  $v$ .

More formally,  $G'$  has two nodes, denoted as  $n(u, v)$  and  $n(v, u)$ , for each (undirected) edge  $(u, v)$  of  $G$  and one node, denoted  $v$ , for each isolated node  $v$  of  $G$ .

For each non-isolated node  $u$  of  $G$ , the nodes of the form  $n(u, v)$  are connected in some cyclic order. To this end, the algorithm maintains a doubly linked list of nodes of  $G'$ , for each node  $u$  of  $G$ .

An insertion of a new edge  $(u, v)$  into  $G$  translates into activating two new nodes  $n(u, v)$  and  $n(v, u)$  in  $G'$ , to connect them with each other and to insert them at an arbitrary position into the cycle of  $u$  and  $v$  respectively. Together, this yields 2 node additions, 2 edge deletions and 5 edge insertions.

A deletion of an edge  $(u, v)$  boils down to the reverse operations: 5 edge deletions, 2 edge insertions and 2 node deactivations.

And obviously, a connectivity query towards  $G$  can just be translated into a connectivity query towards  $G'$ .

Altogether, each operation for  $G$  can be translated into a constant number of operations for  $G'$ . The number of nodes of  $G'$  is linear in the number of edges plus the number of (isolated) nodes of  $G$ . Therefore, the work bounds  $\tilde{\mathcal{O}}(m^{\frac{1}{2}})$  and  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  for  $G'$  translates into a work bound  $\tilde{\mathcal{O}}(m^{\frac{1}{2}})$  and  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  for  $G$ . ◀

The final step towards Theorem 3.1 is to show that the work bounds  $\tilde{\mathcal{O}}(m^{\frac{1}{2}})$  and  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  for maintaining SPANNINGFOREST can be replaced by  $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$  and  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$ , thus showing Proposition 3.4. We use the sparsification technique of [4] which has also been used in [12] to maintain CONNECTIVITY in a parallel setting.

In a nutshell, the approach is to maintain a so-called *sparsification tree*  $\mathcal{S}$ , that is a rooted tree of logarithmic depth in  $n$ , each node  $u$  of which represents a certain subgraph  $G_u$  of  $G$  and carries additional structure. The root represents the whole graph, each leaf represents a subgraph consisting of (at most) one edge, and the graph of each inner node is basically the union of the edge sets of the graphs of its children. The crucial idea is that  $\mathcal{S}$  has an additional *base graph*  $B_u$ , for each tree node, which has a subset of the edges of  $G_u$  of linear size, and has the same connected components (viewed as sets of nodes) as  $G_u$ . Furthermore, the algorithm maintains a spanning forest  $F_u$  of  $B_u$  (and thus for  $G_u$ ), for each tree node  $u$ , using the algorithm with work bound  $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$  or  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  depending on the PRAM model. It has the invariant that, for each inner node  $u$ ,  $B_u$  consists of all edges of the spanning forests  $F_v$ , for all children  $v$  of  $u$ . As this number will be a constant (in fact: 4), the invariant guarantees the linear number of edges of  $B_u$ .

We will see that each change operation can be basically handled by triggering change operations along one path of the tree. Since the base graph of a node at level  $i$  has at most  $\frac{cn}{2^i}$  many edges, for some constant  $c$ , and, the overall work can be bounded by  $\tilde{\mathcal{O}}(n^{\frac{1}{2}} \log n) = \tilde{\mathcal{O}}(n^{\frac{1}{2}})$  and  $\mathcal{O}(n^{\frac{1}{2}+\epsilon'} \log n) = \mathcal{O}(n^{\frac{1}{2}+\epsilon})$ , if  $\epsilon'$  is chosen appropriately.<sup>8</sup>

We next describe the underlying tree structure of  $\mathcal{S}$  in more detail. It relies<sup>9</sup> on a *node partition tree*  $\mathcal{N}(G)$ : it is a binary tree, in which each node is a set  $U$  of nodes of  $G$ . The root is the set  $V$  of all nodes and, for each inner node  $U$  with children  $U_1, U_2$ ,  $U$  is the disjoint union of  $U_1$  and  $U_2$  and the sizes of  $U_1$  and  $U_2$  differ by at most 1. The leaves are the singleton sets. Clearly this tree has depth at most  $\log(n) + 1$ . We emphasise that the partitions are independent of the edge set of  $G$ , they do not need to partition the graph into meaningful clusters.

The edge set of  $G$  and  $\mathcal{N}(G)$  determine the structure of the sparsification tree  $\mathcal{S}$  and its graphs  $G_u$  as follows. For each level  $i$  of  $\mathcal{N}(G)$ ,  $\mathcal{S}$  has one node  $G_u$ , for each pair  $(V_1, V_2)$  of nodes of  $\mathcal{N}(G)$  of level  $i$ . Here,  $V_1 = V_2$  is allowed. The node set of  $G_u$  is  $V_1 \cup V_2$  and

<sup>8</sup> In fact, using the sizes of the base graphs along a path and the “well-behavedness” of  $n^{\frac{1}{2}+\epsilon}$ , one actually gets a  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$  bound, directly.

<sup>9</sup> We remark that for our algorithm it is actually not important how the (potential) edges of the graph  $G$  are exactly partitioned in the sparsification tree, as long as it has constant branching, logarithmic depth and the correspondence between edge sets of nodes and of their children. The definition with the help of the node partition tree is just one concrete way of doing it.

the edges are the edges of  $G$  that connect a node from  $V_1$  with a node from  $V_2$ . If  $V_1 = V_2$ , then  $G_u$  is thus just the subgraph of  $G$  induced by  $V_1$ . If  $U_1$  is the parent of  $V_1$  and  $U_2$  the parent of  $V_2$  in  $\mathcal{N}(G)$ , then the node corresponding to  $(U_1, U_2)$  is the parent of  $(V_1, V_2)$  in  $\mathcal{S}$ . For a leaf  $u$ ,  $G_u$  either has one edge or has no edges, if the edge for the pair  $(x, y)$  of nodes corresponding to  $u$  is not present in  $G$ .

The base graphs  $B_u$  and the spanning forests  $F_u$  can be chosen in any way that is consistent with the invariant that, for each inner node  $u$ ,  $B_u$  consists of all edges of the spanning forests  $F_v$ , for all children  $v$  of  $u$ .

Although our presentation slightly differs from [4, 12], the node partition tree and the sparsification tree are basically the same as there.

Before we present the proof of Proposition 3.4, we state some helpful observations about  $\mathcal{S}$ .

- (1) Each edge  $(x, y)$  of  $G$  occurs exactly in the graphs  $G_u$  along the paths from the leaf with  $(x, y)$  to the root.
- (2) If two nodes  $x, y$  are in the same connected component in  $G_u$ , this also holds in all  $G_v$ , where  $v$  is an ancestor of  $u$ .
- (3) If an edge  $(x, y)$  occurs in some spanning forest  $F_u$ , then it occurs in all spanning forests  $F_v$  on the path from  $u$  to the leaf containing  $(x, y)$ .

**Proof sketch (of Proposition 3.4).** The algorithm maintains a sparsification tree  $\mathcal{S}$  for the graph  $G$ . For each node  $u$  of  $\mathcal{S}$  it maintains a spanning forest  $F_u$  for  $B_u$  (and implicitly, for  $G_u$ ) with the help of the algorithm for SPANNINGFOREST from Proposition 3.3, with  $c = 4$ .

If an edge  $(x, y)$  is inserted to  $G$ , the algorithm checks, for each node  $u$  on the path  $\pi$  from the leaf for  $(x, y)$  to the root, whether  $x$  and  $y$  are in the same connected component of  $T_u$ . From Observation (2) it follows that the nodes  $u$ , for which this is *not* the case constitute some initial segment of  $\pi$ . For all these nodes  $u$ ,  $(x, y)$  is added to  $B_u$  and  $T_u$ . Furthermore, it is added to  $B_v$  of the parent  $v$  of the last node of  $\pi$ .

The deletion of an edge  $(x, y)$  is slightly more complicated. For all nodes  $u$  on the path from the leaf  $v$  with  $(x, y)$  to the root, the algorithm tests in parallel, whether  $(x, y)$  occurs in  $F_u$ . Thanks to Observation (3), the nodes  $v$ , for which this test is positive form an initial segment  $\pi'$  of  $\pi$  up to some node  $w$ . For each of these nodes, the algorithm computes a replacement edge for  $(x, y)$ , if such exists. Thanks to Observation (2), a replacement edge that works for some  $F_v$  is also a replacement edge for all nodes on  $\pi'$  above  $v$ . In particular, all edges  $v$ , for which  $F_v$  has a replacement edge form an upper segment of  $\pi'$  and the replacement edge for the lowest  $F_v$  can be used for all of them. Therefore, after doing the initial test and computing a replacement edge for each forest, constant time and work  $\mathcal{O}((\log n)^2)$  suffice to determine the lowest node  $w$  and its replacement edge  $e$ . Since  $(\log n)^2 = \mathcal{O}(\text{polylog } n)$  and  $(\log n)^\epsilon = \mathcal{O}(n^\epsilon)$ , for each  $\epsilon > 0$ , this work can be neglected. Afterwards, for each node  $u$  of  $\pi'$  above  $w$ ,  $(x, y)$  is deleted from  $B_u$  and  $F_u$  and instead  $e$  is added. In the base graph of the parent of  $w$ , edge  $(x, y)$  is deleted and  $e$  added.

As already explained above, the algorithm applies at most a logarithmic (in  $n$ ) number of times an operation of the algorithm underlying Proposition 3.3, for a base graph, i.e., a graph with  $\mathcal{O}(n)$  edges. The desired work bound  $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$  for arbitrary CRCW PRAMs is thus immediate and by choosing in Proposition 3.3, any  $\epsilon' < \epsilon$  instead of the given  $\epsilon$ , we can establish the desired work bound  $\mathcal{O}(n^{\frac{1}{2}+\epsilon'})$  for common CRCW PRAMs. ◀

## 4 Bipartiteness

In this section, we show that the work bound established for CONNECTIVITY in Section 3 also holds for BIPARTITENESS. In fact, the algorithm will rely on the algorithm of Proposition 3.3.

► **Theorem 4.1.** *There are dynamic parallel constant-time algorithms for BIPARTITENESS with the following work bounds per change or query operation.*

- $\tilde{O}(n^{\frac{1}{2}})$  work on the arbitrary CRCW PRAM model.
- $O(n^{\frac{1}{2}+\epsilon})$  work on the common CRCW PRAM model, for every  $\epsilon > 0$ .

The result follows from an analogous series of statements, as for CONNECTIVITY (or SPANNINGFOREST, for that matter).

► **Proposition 4.2.** *There are dynamic parallel constant time algorithms for the special case of BIPARTITENESS, where the maximum degree of the graph never exceeds 3 with the following work bounds per change or query operation.*

- $\tilde{O}(m^{\frac{1}{2}})$  on the arbitrary CRCW PRAM model.
- $O(m^{\frac{1}{2}+\epsilon})$  on the common CRCW PRAM model, for every  $\epsilon > 0$ .

► **Proposition 4.3.** *If BIPARTITENESS can be maintained in parallel constant time on a CRCW PRAM with the work bounds of Proposition 4.2 per change or query operation, for any  $\epsilon > 0$ , for graphs with maximum degree 3, it can be maintained with the same bounds for general graphs with the provision that they never have more than  $cn$  edges, for some constant  $c$ .*

► **Proposition 4.4.** *If BIPARTITENESS can be maintained in parallel constant time on a CRCW PRAM with the work bounds of Proposition 4.2 per change or query operation, then it can also be maintained with  $\tilde{O}(n^{\frac{1}{2}})$  work per change or query operation on the common CRCW model and with  $O(n^{\frac{1}{2}+\epsilon})$  work per change or query operation on the arbitrary CRCW model.*

For an undirected graph  $G = (V, E)$ , we write  $G^{(2)}$  for the graph<sup>10</sup>  $(V, E^{(2)})$ , where a pair  $(u, v)$  of nodes is in  $E^{(2)}$ , if they are connected by a path of length exactly 2 in  $G$ .

Bipartiteness of a graph  $G$  can be characterised in the following way by the numbers of connected components of  $G$  and  $G^{(2)}$ .

► **Lemma 4.5.** *An undirected graph  $G$  is bipartite, if and only if the number of connected components of  $G^{(2)}$  is twice the number of connected components of  $G$ .*

**Proof.** It suffices to show that a connected graph  $G$  is bipartite if and only if  $G^{(2)}$  has 2 connected components.

Let us assume first that  $G$  is bipartite and let the nodes of  $G$  be coloured with black or yellow such that no two nodes of the same color are connected by an edge. Clearly, each pair of nodes of the same color is connected by a path of even length in  $G$  and is therefore in the same connected component in  $G^{(2)}$ .

Towards a contradiction, let us assume that  $G^{(2)}$  is connected. Then there must be a yellow node  $u$  and a black node  $v$  that are connected by an edge in  $G^{(2)}$ . Therefore, there must be a node  $w$ , such that  $(u, w)$  and  $(w, v)$  are edges in  $G$ . But  $w$  can neither be black nor yellow, the desired contradiction.

Let us now assume that  $G$  is not bipartite and let  $C$  be a cycle of  $G$  of odd length. Then all pairs of nodes of  $C$  are connected by paths of even length and therefore all nodes of  $C$  are in the same connected component of  $G^{(2)}$ . But clearly, each other node of  $G$  is connected by a path of even length to *some* node of  $C$  and thus  $G^{(2)}$  is connected. ◀

---

<sup>10</sup>  $G^{(2)}$  should not be confused with the square  $G^2$  of  $G$ , where edges are induced by paths of length at most 2.

With the help of Lemma 4.5, it is now easy to find an algorithm for BIPARTITENESS for graphs of degree at most 3.

**Proof (of Proposition 4.2).** To maintain bipartiteness for a graph  $G$  of maximum degree 3, the algorithm maintains two instances of SPANNINGFOREST, one for  $G$ , and one for  $G^{(2)}$ . It answers that  $G$  is bipartite, whenever the number of connected components of  $G^{(2)}$  is twice the number of connected components of  $G$ .

An edge insertion in  $G$  results in at most 6 edge insertions in  $G^{(2)}$ , and likewise for edge deletions. Furthermore, the number of edges of  $G^{(2)}$  is at most  $3m$ , if  $m$  is the number of edges of  $G$ . Therefore, BIPARTITENESS can be maintained in parallel constant time with work  $\tilde{O}(m^{\frac{1}{2}})$  or rather  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$ , thanks to Proposition 3.3. ◀

Next we lift the bound to graphs of unbounded degree with the help of a bipartiteness preserving reduction.

**Proof (of Proposition 4.3).** To maintain bipartiteness of graph  $G$ , the algorithm again maintains bipartiteness for a graph  $G'$  of maximal degree 3, such that  $G$  is bipartite if and only if  $G'$  is bipartite. The graph  $G'$  results from  $G$  by applying the following replacement step, consecutively to all (original) nodes of  $G$ .

A node  $u$  of degree  $d > 1$  is replaced by a cycle  $u_1, u'_1, u_2, \dots, u'_d, u_1$  with  $2d$  nodes. Each node  $u_i$  is connected to a neighbour of  $u$  by an edge. It is easy to see that any path that connects two neighbours of  $u$  and uses intermediate nodes of the new cycle has even length. The construction therefore preserves bipartiteness. Furthermore, each node in  $G'$  has degree at most 3 and the number of edges of  $G'$  is at most 6 times the number of edges of  $G$ . Finally, each edge insertion or deletion in  $G$  triggers at most 5 edge insertions or deletions in  $G'$ . ◀

The final step from work  $\tilde{O}(m^{\frac{1}{2}})$  to  $\tilde{O}(n^{\frac{1}{2}})$  and work  $\mathcal{O}(m^{\frac{1}{2}+\epsilon})$  to  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$  again uses sparsification. In fact, it uses the same kind of sparsification tree as the proof of Proposition 3.4. The crucial observation is that if a graph  $G$  is not bipartite, it has a base graph in its sparsification tree that is not bipartite.

► **Lemma 4.6.** *Let  $G$  be an undirected graph and  $\mathcal{S}$  a sparsification tree for  $G$ . Then  $G$  is bipartite if and only if all base graphs in  $\mathcal{S}$  are bipartite.*

**Proof.** Since each base graph of  $\mathcal{S}$  is a subgraph of  $G$ , the “only if” implication is trivial.

To show the “if” implication, let  $G$  be a non-bipartite graph. Since the graph  $G_r$  for the root  $r$  of  $\mathcal{S}$  is non-bipartite, but all graphs  $G_v$  for leaves  $v$  of  $\mathcal{S}$  are bipartite, there must be a node  $u$ , such that  $G_u$  is non-bipartite, but all graphs  $G_w$ , for children  $w$  of  $u$ , are bipartite. We claim that the base graph  $B_u$  is non-bipartite.

Indeed, let  $C$  be some cycle of odd length in  $G_u$ . By construction of  $\mathcal{S}$ , each edge  $(x, y)$  of  $C$  occurs in some graph  $G_w$ , where  $w$  is a child of  $u$ . Therefore,  $x$  and  $y$  are in the same connected component of  $G_w$  and there must be a path between  $x$  and  $y$  in the spanning forest  $F_w$ . Since  $G_w$  is bipartite and there is an edge between  $x$  and  $y$ , the length of this path must be odd. By definition, all edges of this path are in  $B_u$ . Since this holds for every edge of  $C$ , there exists a closed path in  $B_u$ , consisting of an odd number of paths of odd length. This implies that  $B_u$  has a cycle of odd length and is therefore not bipartite. ◀

Now we are prepared to give the proof of Proposition 4.4 and thus complete the proof of Theorem 4.1.

**Proof (of Proposition 4.4).** Just like for Proposition 3.4, the algorithm maintains a sparsification tree  $\mathcal{S}$  for the graph  $G$ . For each node  $u$  of  $\mathcal{S}$  it maintains whether  $B_u$  is bipartite with the algorithm resulting from Proposition 4.2 and Proposition 4.3. This is possible with work bounds  $\tilde{\mathcal{O}}(n^{\frac{1}{2}})$  and  $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$  per change operation, just as for Proposition 3.4.

On top of that, the algorithm maintains, for each node  $u$  of  $\mathcal{S}$ , a flag, signalling whether all base graphs in the tree induced by  $u$  are bipartite. These flags can be maintained in a straightforward fashion with work  $\mathcal{O}(\log n)$ . The bipartiteness status of  $G$  can then be inferred from the flag of the root of  $\mathcal{S}$ , thanks to Lemma 4.6. ◀

## 5 Conclusion

This paper was motivated by the goal to find graph problems whose sublinear sequential dynamic complexity carries over to sublinear work of a dynamic parallel constant time algorithm. In future work it has to be seen whether the faster algorithm from [1] can be translated equally well. Another challenge is to find a dynamic parallel constant time algorithm for the reachability problem in directed graphs. The upper work bound of the algorithm stemming from [2] is roughly  $\mathcal{O}(n^{12})$ . Another interesting question is whether the algorithm for BIPARTITENESS can be adapted so that it also yields a 2-colouring of the graph.

---

### References

- 1 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 2020. doi:10.1109/FOCS46700.2020.00111.
- 2 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.
- 3 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages*, pages 295–308, 1993.
- 4 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.
- 5 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. doi:10.1137/0214055.
- 6 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms (invited talk). In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 1:1–1:47, 2022. doi:10.4230/LIPIcs.SAND.2022.1.
- 7 Scott Huddleston and Kurt Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17:157–184, 1982. doi:10.1007/BF00288968.
- 8 N. Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18:625–638, 1989.
- 9 Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 49–58, 2019. doi:10.1145/3323165.3323202.
- 10 Joseph F. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
- 11 Jens Keppeler, Thomas Schwentick, and Christopher Spinrath. Work-efficient query evaluation with prams. In Floris Geerts and Brecht Vandevoort, editors, *26th International Conference on Database Theory (ICDT)*, pages 16:1–16:20, 2023. doi:10.4230/LIPIcs.ICDT.2023.16.

- 12 Tsvi Kopelowitz, Ely Porat, and Yair Rosenmutter. Improved worst-case deterministic parallel dynamic minimum spanning forest. In Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 333–341, 2018. doi:10.1145/3210377.3210403.
- 13 Sushant Patnaik and Neil Immerman. Dyn-FO: A Parallel, Dynamic Complexity Class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.
- 14 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Sketches of dynamic complexity. *SIGMOD Rec.*, 49(2):18–29, 2020. doi:10.1145/3442322.3442325.
- 15 Peter van Emde Boas. Machine models and simulation. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier and MIT Press, 1990.



# On the Work of Dynamic Constant-Time Parallel Algorithms for Regular Tree Languages and Context-Free Languages

Jonas Schmidt ✉

TU Dortmund University, Germany

Thomas Schwentick ✉

TU Dortmund University, Germany

Jennifer Todtenhoefer ✉

TU Dortmund University, Germany

---

## Abstract

---

Previous work on Dynamic Complexity has established that there exist dynamic constant-time parallel algorithms for regular tree languages and context-free languages under label or symbol changes. However, these algorithms were not developed with the goal to minimise work (or, equivalently, the number of processors). In fact, their inspection yields the work bounds  $\mathcal{O}(n^2)$  and  $\mathcal{O}(n^7)$  per change operation, respectively.

In this paper, dynamic algorithms for regular tree languages are proposed that generalise the previous algorithms in that they allow unbounded node rank and leaf insertions, while improving the work bound from  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n^\epsilon)$ , for arbitrary  $\epsilon > 0$ .

For context-free languages, algorithms with better work bounds (compared with  $\mathcal{O}(n^7)$ ) for restricted classes are proposed: for every  $\epsilon > 0$  there are such algorithms for deterministic context-free languages with work bound  $\mathcal{O}(n^{3+\epsilon})$  and for visibly pushdown languages with work bound  $\mathcal{O}(n^{2+\epsilon})$ .

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

**Keywords and phrases** Dynamic complexity, work, parallel constant time

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.81

**Related Version** *Full Version*: <http://arxiv.org/abs/2307.10131>

**Acknowledgements** We are grateful to Jens Keppeler and Christopher Spinrath for careful proof reading.

## 1 Introduction

It has been known for many years that regular and context-free string languages and regular tree languages are maintainable under symbol changes by means of dynamic algorithms that are specified by formulas of first-order logic, that is, in the dynamic class DynFO [10, 7]. It is also well-known that such specifications can be turned into parallel algorithms for the CRCW PRAM model that require only constant time [8] and polynomially many processors.

However, an “automatic” translation of a “dynamic program” of the DynFO setting usually yields a parallel algorithm with large work, i.e., overall number of operations performed by all processors.<sup>1</sup> In the case of regular languages, the dynamic program sketched in [10] has a polynomial work bound, in which the exponent of the polynomial depends on the number of states of a DFA for the language at hand. The dynamic program given in [7] has quadratic work.

---

<sup>1</sup> We note that in the context of constant-time parallel algorithms work is within a constant factor of the number of processors.



Only recently a line of research has started that tries to determine, how efficient such constant-time dynamic algorithms can be made with respect to their work. It turned out that regular languages can be maintained with work  $\mathcal{O}(n^\epsilon)$ , for every  $\epsilon > 0$  [11], even under polylogarithmic numbers of changes [12], and even with logarithmic work for star-free languages under single changes [11] and polylogarithmic work under polylogarithmic changes [12].

For context-free languages the situation is much less clear. The dynamic algorithms resulting from [7] have an  $\mathcal{O}(n^7)$  upper work bound. In [11] it was shown that the Dyck-1 language, i.e., the set of well-bracketed strings with one bracket type, can be maintained with work  $\mathcal{O}((\log n)^3)$  and that Dyck- $k$  languages can be maintained with work  $\mathcal{O}(n \log n)$ . Here, the factor  $n$  is due to the problem to test equality of two substrings of a string.

Most of these results also hold for the query that asks for membership of a substring in the given language. For Dyck languages the upper bounds for substring queries are worse than the bounds for membership queries: for every  $\epsilon > 0$  there exist algorithms for Dyck-1 and Dyck- $k$  languages with work bounds  $\mathcal{O}(n^\epsilon)$  and  $\mathcal{O}(n^{1+\epsilon})$ , respectively.

It was also shown in [11] that there is some context-free language that can be maintained in constant time with work  $\mathcal{O}(n^{\omega-1-\epsilon})$ , for any  $\epsilon > 0$ , only if the  $k$ -Clique conjecture [1] fails. Here,  $\omega$  is the matrix multiplication exponent, which is known to be smaller than 2.373 and conjectured by some to be exactly two according to [14].

In this paper, we pursue two natural research directions.

**Regular tree languages.** We first extend the results on regular string languages to regular tree languages. On one hand, this requires to adapt techniques from strings to trees. On the other hand, trees offer additional types of change operations beyond label changes that might change the structure of the tree. More concretely, besides label changes we study insertions of new leaves and show that the favourable bounds of [11] for regular string languages still hold. This is the main contribution of this paper. Our algorithms rely on a hierarchical partition of the tree of constant depth. The main technical challenge is to maintain such a partition hierarchy under insertion<sup>2</sup> of leaves.

**Subclasses of context-free languages.** We tried to improve on the  $\mathcal{O}(n^7)$  upper work bound for context-free languages, but did not succeed yet. The other goal of this paper is thus to find better bounds for important subclasses of the context-free languages: deterministic context-free languages and visibly pushdown languages. We show that, for each  $\epsilon > 0$ , there are constant-time dynamic algorithms with work  $\mathcal{O}(n^{3+\epsilon})$  for deterministic context-free languages and  $\mathcal{O}(n^{2+\epsilon})$  for visibly pushdown languages. Here, the main challenge is to carefully apply the technique from [11] that allows to store information for only  $\mathcal{O}(n^\epsilon)$  as opposed to  $n$  different values for some parameters. For more restricted change operations, the algorithm for regular tree languages yields an  $\mathcal{O}(n^\epsilon)$  work algorithm for visibly pushdown languages.

**Structure of the paper.** We explain the framework in Section 2, and present the results on regular tree languages and context-free string languages in Sections 3 and 4, respectively. Almost all proofs are delegated to the full version of this paper.

---

<sup>2</sup> For simplicity, we only consider insertions of leaves, but deletions can be handled in a straightforward manner, as discussed in Section 2.

**Related work.** In [12], parallel dynamic algorithms for regular string languages under bulk changes were studied. It was shown that membership in a regular language can be maintained, for every  $\epsilon > 0$ , in constant time with work  $\mathcal{O}(n^\epsilon)$ , even if a polylogarithmic number of changes can be applied in one change operation. If the language is star-free, polylogarithmic work suffices. The paper also shows that for regular languages that are not star-free, polylogarithmic work does *not* suffice.

Maintaining regular languages of trees under label changes has also been studied in the context of enumeration algorithms (for non-Boolean queries) [3]. The dynamic parallel algorithms of [11] partially rely on dynamic sequential algorithms, especially [6].

## 2 Preliminaries

**Trees and regular tree languages.** We consider ordered, unranked trees  $t$ , which we represent as tuples  $(V, r, c, \text{label})$ , where  $V$  is a finite set of nodes,  $r \in V$  is the root,  $c : V \times \mathbb{N} \rightarrow V$  is a function, such that  $c(u, i)$  yields the  $i$ -th child of  $u$ , and  $\text{label} : V \rightarrow \Sigma$  is a function that assigns a label to every node.

We denote the set of unranked trees over an alphabet  $\Sigma$  as  $T(\Sigma)$ . The terms *subtree*, *subforest*, *sibling*, *ancestor*, *descendant*, *depth* and *height* of nodes are defined as usual. A node that has no child is called a *leaf*. A *forest* is a sequence of trees.

Let  $\preceq$  denote the order on siblings, i.e.,  $u \prec v$  denotes that  $u$  is a sibling to the left of  $v$ . We write  $u \preceq v$  if  $u \prec v$  or  $u = v$  holds.

By  $t^v$  we denote the subtree of  $t$  induced by node  $v$ . For sibling nodes  $u \prec v$ , we write  ${}^u t^v$  for the subforest of the tree  $t$ , induced by the sequence  $u, \dots, v$ . If  $w$  is a node in  $t^v$ , then  $t_w^v$  denotes the subtree consisting of  $t^v$  without  $t^w$ . Analogously, for  ${}^u t_w^v$ .

Our definition of tree automata is inspired from hedge automata in the TaTa book [5], slightly adapted for our needs.

► **Definition 2.1.** A deterministic finite (bottom-up) tree automaton (DTA) over an alphabet  $\Sigma$  is a tuple  $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, Q_f, \delta, \mathcal{A})$  where  $Q_{\mathcal{B}}$  is a finite set of states,  $Q_f \subseteq Q_{\mathcal{B}}$  is a set of final states,  $\mathcal{A} = (Q_{\mathcal{A}}, Q_{\mathcal{B}}, \delta_{\mathcal{A}}, s)$  is a DFA over alphabet  $Q_{\mathcal{B}}$  (without final states) and  $\delta : Q_{\mathcal{A}} \times \Sigma \rightarrow Q_{\mathcal{B}}$  maps pairs  $(p, \sigma)$ , where  $p$  is a state of  $\mathcal{A}$  and  $\sigma \in \Sigma$ , to states of  $\mathcal{B}$ .

We refer to states from  $Q_{\mathcal{B}}$  as  $\mathcal{B}$ -states and typically denote them by the letter  $q$ . Likewise states from  $Q_{\mathcal{A}}$  are called  $\mathcal{A}$ -states and denoted by  $p$ . We note that we do not need a set of accepting states for  $\mathcal{A}$ , since its final states are fed into  $\delta$ .

The semantics of DTAs is defined as follows.

For each tree  $t \in T(\Sigma)$ , there is a unique *run* of  $\mathcal{B}$  on  $t$ , that is, a unary function  $\rho_t$  that assigns a  $\mathcal{B}$ -state to each node in  $V$ . It can be defined in a bottom-up fashion, as follows. For each node  $v \in V$  with label  $\sigma$  and children  $u_1, \dots, u_\ell$ ,  $\rho_t(v)$  is the  $\mathcal{B}$ -state  $\delta(\delta_{\mathcal{A}}^*(s, \rho_t(u_1) \cdots \rho_t(u_\ell)), \sigma)$ . That is, the state of a node  $v$  with label  $\sigma$  is determined by  $\delta(p, \sigma)$ , where  $p$  is the final  $\mathcal{A}$ -state that  $\mathcal{A}$  assumes when reading the sequence of states of  $v$ 's children, starting from the initial state  $s$ . In particular, if  $v$  is a leaf with label  $\sigma$ , its  $\mathcal{B}$ -state is  $\delta(s, \sigma)$ .

A tree  $t$  is accepted by the DTA  $\mathcal{B}$  if  $\rho_t(r) \in Q_f$  holds for the root  $r$  of  $t$ . We denote the language of all trees accepted by  $\mathcal{B}$  as  $L(\mathcal{B})$ . We call the languages decided by DTAs *regular*.

**Strings and context-free languages.** Strings  $w$  are finite sequences of symbols from an alphabet  $\Sigma$ . By  $w[i]$  we denote the  $i$ -th symbol of  $w$  and by  $w[i, j]$  we denote the substring from position  $i$  to  $j$ . We denote the empty string by  $\lambda$ , since  $\epsilon$  has a different purpose in this paper. We use standard notation for context-free languages and pushdown automata, to be found in the full version of this paper.

**Dynamic algorithmic problems.** In this paper, we view a dynamic (algorithmic) problem basically as the interface of a data type: that is, there is a collection of operations by which some object can be initialised, changed, and queried. A *dynamic algorithm* is then a collection of algorithms, one for each operation. We consider two main dynamic problems in this paper, for regular tree languages and context-free languages.

For each regular tree language  $L$ , the algorithmic problem  $\text{REGTREE}(L)$  maintains a labelled tree  $T$  and has the following operations.

- $\text{Init}(T, r, \sigma)$  yields an initial labelled tree object  $T$  and returns in  $r$  a node id for its root, which is labelled by  $\sigma$ ;
- $\text{Relabel}(T, u, \sigma)$  changes the label of node  $u$  in  $T$  into  $\sigma$ ;
- $\text{AddChild}(T, u, v, \sigma)$  adds a new child with label  $\sigma$  behind the last child of node  $u$  and returns its id in  $v$ ;
- $\text{Query}(T, v)$  returns true if and only if the subtree of  $T$  rooted at  $v$  is in  $L$ .

We refer to the restricted problem without the operation  $\text{AddChild}$  as  $\text{REGTREE}^-$ . For this data type, we assume that the computation starts from an initial non-trivial tree and that the auxiliary data for that tree is given initially.

For each context-free language  $L$ , the algorithmic problem  $\text{CFL}(L)$  maintains a string  $w$  and has the following operations.

- $\text{Init}(w)$  yields an initial string object  $w$  with an empty string;
- $\text{Relabel}(w, i, \sigma)$  changes the label at position  $i$  of  $w$  into  $\sigma$ ;
- $\text{InsertPositionBefore}(w, i, \sigma)$  and  $\text{InsertPositionAfter}(w, i, \sigma)$  insert a new position with symbol  $\sigma$  before or after the current position  $i$ , respectively;
- $\text{Query}(w, i, j)$  returns true if and only if the substring  $w[i, j]$  is in  $L$ .

Readers may wonder, why these dynamic problems do not have operations that delete nodes of a tree or positions in a string. This is partially to keep the setting simple and partially because node labels and symbols offer easy ways to simulate deletion by extending the alphabet with a symbol  $\sqcup$  that indicates an object that should be ignored. E.g., if  $\delta_A(p, \sqcup) = p$ , for every state  $p$  of the horizontal DFA of a DTA, then the label  $\sqcup$  at a node  $u$  effectively deletes the whole subtree induced by  $u$  for the purpose of membership in  $L(\mathcal{B})$ . Similarly, a CFL might have a neutral symbol or even a pair  $(\sqcup, )_{\sqcup}$  of “erasing” brackets that make the PDA ignore the substring between  $(\sqcup$  and  $)_{\sqcup}$ .

For  $\text{REGTREE}(L)$  and  $\text{CFL}(L)$ , the  $\text{Init}$  operation is possible in constant sequential time and will not be considered in detail.

Throughout this paper,  $n$  will denote an upper bound of the size of the structure at hand (number of nodes of a tree or positions of a string) that is linear in that size, but changes only infrequently. More precisely, the number of nodes of a tree or the length of the string will always be between  $\frac{1}{4}n$  and  $n$ . Whenever the size of the structure grows beyond  $\frac{1}{2}n$ , the data structure will be prepared for structures of size up to  $2n$  and, once this is done,  $n$  will be doubled. Since the size of the structure is always  $\theta(n)$  all bounds in  $n$  also hold with respect to the size of the structure.

**Parallel Random Access Machines (PRAMs).** A *parallel random access machine* (PRAM) consists of a number of processors that work in parallel and use a shared memory. The memory is comprised of memory cells which can be accessed by a processor in  $\mathcal{O}(1)$  time. Furthermore, we assume that simple arithmetic and bitwise operations, including addition, can be done in  $\mathcal{O}(1)$  time by a processor. We mostly use the Concurrent-Read Concurrent-Write model (CRCW PRAM), i.e. processors are allowed to read and write concurrently from and to the same memory location. More precisely, we assume the *common* PRAM model:

several processors can concurrently write into the same memory location, only if all of them write the same value. We also mention the Exclusive-Read Exclusive-Write model (EREW PRAM), where concurrent access is not allowed. The work of a PRAM computation is the sum of the number of all computation steps of all processors made during the computation. We define the space  $s$  required by a PRAM computation as the maximal index of any memory cell accessed during the computation. We refer to [9] for more details on PRAMs and to [13, Section 2.2.3] for a discussion of alternative space measures.

The main feature of the common CRCW model relevant for our algorithms that separates it from the EREW model is that it allows to compute the minimum or maximum value of an array of size  $n$  in constant time (with work  $\mathcal{O}(n^{1+\epsilon})$ ) which is shown in another paper at MFCS 2023.<sup>3</sup>

For simplicity, we assume that even if the size bound  $n$  grows, a number in the range  $[0, n]$  can still be stored in one memory cell. This assumption is justified, since addition of larger numbers  $N$  can still be done in constant time and polylogarithmic work on a CRCW PRAM. Additionally, we assume that the number of processors always depends on the current size bound  $n$ . Hence, the number of processors increases with growing  $n$  which allows us to use the PRAM model with growing structures.

We describe our PRAM algorithms on an abstract level and do not exactly specify how processors are assigned to data. Whenever an algorithm does something in parallel for a set of objects, these objects can be assigned to a bunch of processors with the help of some underlying array. This is relatively straightforward for strings and substrings and the data structures used in Section 4. In Section 3, it is usually based on zone records and their underlying partition records.

### 3 Maintaining regular tree languages

In this section, we present our results on maintaining regular tree languages under various change operations. We will first consider only operations that change node labels, but do not change the shape of the given tree. A very simple dynamic algorithm with work  $\mathcal{O}(n^2)$  is presented in the full version of this paper. We sketch its main idea and how it can be improved to  $\mathcal{O}(n^\epsilon)$  work per change operation by using a *partition hierarchy* in Subsection 3.1. These algorithms even work on the EREW PRAM model.

Afterwards, in Subsection 3.2, we also consider an operation that can change the tree structure: adding a leaf to the tree. Here, the challenge is to maintain the hierarchical structure that we used before to achieve work  $\mathcal{O}(n^\epsilon)$  per change operation. It turns out that maintaining this structure is possible without a significant increase of work, that is, maintaining membership under these additional operations is still possible with work  $\mathcal{O}(n^\epsilon)$  per change operation.

#### 3.1 Label changes: a work-efficient dynamic program

In this section, we describe how membership in a regular tree language can be maintained under label changes, in a work efficient way.

---

<sup>3</sup> Jonas Schmidt, Thomas Schwentick. Dynamic constant time parallel graph algorithms with sub-linear work.

► **Proposition 3.1.** *For each  $\epsilon > 0$  and each regular tree language  $L$ , there is a parallel constant time dynamic algorithm for  $\text{REGTREE}^-(L)$  with work  $\mathcal{O}(n^\epsilon)$  on an EREW PRAM. The Query operation can actually be answered with constant work.*

We start by briefly sketching the  $\mathcal{O}(n^2)$  work algorithm that is given in the full version of this paper. The algorithm basically combines the dynamic programs for regular string languages and binary regular tree languages from [7]. For regular string languages, the program from [7] stores the behaviour of a DFA for the input word  $w$  by maintaining information of the form “if the run of the DFA starts at position  $i$  of  $w$  and state  $p$ , then it reaches state  $q$  at position  $j$ ” for all states  $p, q$  and substrings  $w[i, j]$ . After a label change at a position  $\ell$ , this information can be constructed by combining the behaviour of the DFA on the intervals  $w[i, \ell - 1]$  and  $w[\ell + 1, j]$  with the transitions induced by the new label at position  $\ell$ .

The dynamic program for (binary) regular tree languages from [7] follows a similar idea and stores the behaviour of a (binary) bottom-up tree automaton by maintaining information of the form “if  $v$  gets assigned state  $q$ , then  $u$  gets assigned state  $p$  by the tree automaton” for all states  $p, q$  and all nodes  $v, u$ , where  $v$  is a descendant of  $u$ .

Both programs induce algorithms with  $\mathcal{O}(n^2)$  work bounds. Towards a  $\mathcal{O}(n^2)$  work algorithm for unranked tree languages, the two dynamic programs can be combined into an algorithm that mainly stores the following *automata functions* for a fixed DTA  $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, Q_f, \delta, \mathcal{A})$  for  $L$ , with DFA  $\mathcal{A} = (Q_{\mathcal{A}}, Q_{\mathcal{B}}, \delta_{\mathcal{A}}, s)$ :

- The ternary function  $\mathcal{B}_t : Q_{\mathcal{B}} \times V \times V \mapsto Q_{\mathcal{B}}$  maps each triple  $(q, u, v)$  of a state  $q \in Q_{\mathcal{B}}$  and nodes of  $t$ , where  $u$  is a proper ancestor of  $v$ , to the state that the run of  $\mathcal{B}$  on  $t_v^u$  takes at  $u$ , with the provision that the state at  $v$  is  $q$ .
- The ternary function  $\mathcal{A}_t : Q_{\mathcal{A}} \times V \times V \mapsto Q_{\mathcal{A}}$  maps each triple  $(p, u, v)$  of a state  $p \in Q_{\mathcal{A}}$  and nodes of  $t$ , where  $u \prec v$  are siblings, to the state that the run of  $\mathcal{A}$  on  $u, \dots, v$ , starting from state  $p$ , takes after  $v$ .

Every single function value can be updated in constant sequential time, as stated in the following lemma. This leads to a quadratic work bound since there are quadratically many tuples to be updated in parallel.

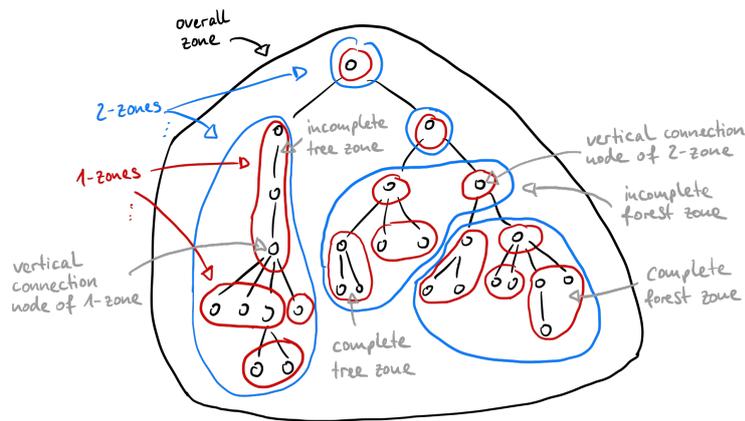
► **Lemma 3.2.** *After a Relabel operation, single values  $\mathcal{A}_t(p, u, x)$  and  $\mathcal{B}_t(q, u, x)$  can be updated by a sequential algorithm in constant time.*

Some information about the shape of the tree is required, which we refer to as *basic tree functions*. For more details we refer to the full version of this paper. However, as label changes cannot change the shape of the tree, this information does not need to be updated and can be assumed as precomputed.

To lower the work bound the basic idea now is to store the automata functions not for *all* possible arguments, but for a small subset of *special* arguments that allow the computation of function values for *arbitrary* arguments in constant time with constant work.

In [11], this idea was applied to the  $\mathcal{O}(n^2)$  work program for regular string languages. A constant-depth hierarchy of intervals was defined by repeatedly partitioning intervals into  $\mathcal{O}(n^\theta)$  subintervals, for some  $\theta > 0$ . This hierarchy allowed to define *special* intervals such that any update only affects  $\mathcal{O}(n^\epsilon)$  intervals and function values of arbitrary intervals can be computed in constant time with constant work.

We transfer this idea to the case of unranked tree languages by partitioning the tree into  $\mathcal{O}(n^\theta)$  *zones*, each of which is partitioned into further  $\mathcal{O}(n^\theta)$  zones and so on until, after a constant number of refinements, we arrive at zones of size  $\mathcal{O}(n^\theta)$ . Here,  $\theta > 0$  is a constant that will be chosen later. It will always be chosen such that  $h = \frac{1}{\theta}$  is an integer.



■ **Figure 1** Example of a  $(1, \frac{1}{3})$ -bounded partition hierarchy.

Before we define this partition hierarchy more precisely, we first define zones and show that they can always be partitioned in a way that guarantees certain number and size constraints.

- **Definition 3.3.** A *zone* is a set  $S$  of nodes with the following properties:
- $S$  is a proper subforest of  $t$ ,
  - for every  $v \in S$  it holds that either no or all children are in  $S$ , and
  - there exists at most one node  $v_S$  in  $S$ , whose children are not in  $S$ . The node  $v_S$  is called the *vertical connection node* of  $S$ .

We call a zone a *tree zone* if it consists of only one sub-tree of  $t$  and a *non-tree zone* otherwise. We call a zone *incomplete* if it has a vertical connection node and *complete*, otherwise. There are thus four different types of zones which can be written, with the notation introduced in Section 2, as follows: complete tree zones  $t^v$ , complete non-tree zones  ${}^u t^v$ , incomplete tree zones  $t_w^v$ , and incomplete non-tree zones  ${}^u t_w^v$ . Depending on the type, zones can therefore be represented by one to three “important nodes”. The overall tree can be seen as the zone  $t^r$ , where  $r$  is its root.

From now on, we always assume that  $n$  is as in Section 2, some  $\theta > 0$  is fixed, and that  $h = \frac{1}{\theta}$  is an integer.

We call a zone of  $t$  with at most  $n^{\theta \ell}$  nodes an  $\ell$ -zone. The tree  $t$  itself constitutes a  $h$ -zone, to which we will refer to as the *overall zone*.

We next define *partition hierarchies* formally. More precisely, for every  $\ell \geq 2$ , we define partition hierarchies of height  $\ell$  for  $\ell$ -zones as follows. If  $S$  is a 2-zone and  $S_1, \dots, S_k$  are 1-zones that constitute a partition of  $S$ , then  $(S, \{S_1, \dots, S_k\})$  is a partition hierarchy of height 2 for  $S$ . If  $S$  is an  $(\ell + 1)$ -zone,  $\{S_1, \dots, S_k\}$  is a partition of  $S$  into  $\ell$ -zones, and for each  $j$ ,  $H_j$  is a partition hierarchy of height  $i$  for  $S_j$ , then  $(S, \{H_1, \dots, H_k\})$  is a partition hierarchy of height  $\ell + 1$  for  $S$ . A partition hierarchy of height  $h$  of the zone consisting of  $t$  is called a partition hierarchy of  $t$ .

An example of a  $(1, \frac{1}{3})$ -bounded partition hierarchy is given in Figure 1.

We often call a zone  $S'$  that occurs at some level  $i < \ell$  within the partition hierarchy of a zone  $S$  of some level  $\ell$  a *component zone*. If  $S'$  has level  $\ell - 1$  we also call it a *sub-zone* of  $S$ .

We call a partition hierarchy  $H$   $(c, \theta)$ -bounded, constants  $c$  and  $\theta > 0$ , if each partition of a zone consists of at most  $cn^\theta$  nodes.

Our next aim is to prove that  $(10, \theta)$ -bounded partition hierarchies actually exist. To this end, we prove the following lemma. It is similar to [4, Lemma 3], but adapted to our context, which requires a hierarchy of constant depth and a certain homogeneity regarding children of vertical connection nodes.

► **Lemma 3.4.** *Let  $m \geq 2$  be a number and  $S$  a zone with more than  $m$  nodes. Then  $S$  can be partitioned into at most five zones, one of which has at least  $\frac{1}{2}m$  and at most  $m$  nodes.*

This lemma immediately yields the existence of  $(10, \theta)$ -bounded partition hierarchies.

► **Proposition 3.5.** *For each  $\theta > 0$ , each tree  $t$  has some  $(10, \theta)$ -bounded partition hierarchy.*

We now explain in more detail, which information about the behaviour of  $\mathcal{A}$  and  $\mathcal{B}$  is stored by the work-efficient algorithm.

Function values for the ternary functions are stored only for so-called special pairs of nodes, which we define next. Special pairs of nodes are always defined in the context of some zone  $S$  of a partition hierarchy. In the following, we denote, for a zone  $S$  of a level  $\ell \geq 2$  its set of sub-zones of level  $\ell - 1$  by  $T$ .

- Any pair of siblings  $u \prec v$  in a zone  $S$  of level 1 is a *special horizontal pair*. A pair of siblings  $u \prec v$  in a complete zone  $S$  of level  $\ell \geq 2$  is a *special horizontal pair*, if  $u$  is a left boundary of some zone in  $T$  and  $v$  is a right boundary of some zone in  $T$ . However, if  $S$  is incomplete and there is an ancestor  $w'$  of the lower boundary  $w$  with  $u \preceq w' \preceq v$ , then, instead of  $(u, v)$ , there are two special pairs:  $(u, \text{LEFT-SIBLING}(w'))$  and  $(\text{RIGHT-SIBLING}(w'), v)$ .
- Any pair of nodes  $u, v$  in some zone  $S$  of level 1 is a *special vertical pair*, if  $v$  is an ancestor of  $u$ . A pair of nodes  $u, v$  in some zone  $S$  of level  $\ell \geq 2$  is a *special vertical pair*, if  $v$  is an ancestor of  $u$ ,  $v$  is an upper or lower boundary of some zone in  $T$  and  $u$  is a lower boundary of some zone in  $T$ . However, if  $S$  is incomplete with lower boundary  $w$  and  $w' := \text{LCA}(w, u)$  is strictly above  $u$  and below or equal to  $v$ , then, instead of  $(u, v)$ , there are two special pairs:  $(u, \text{ANC-CHILD}(w', u))$  and  $(w', v)$ . Here LCA determines the least common ancestor and ANC-CHILD the child of  $w'$  that is an ancestor of  $u$ .

The algorithm stores  $\mathcal{A}_t(p, u, v)$  for each state  $p$  of  $\mathcal{A}$  and each special horizontal pair  $u, v$ . Furthermore, it stores  $\mathcal{B}_t(q, u, v)$ , for each state  $q$  of  $\mathcal{B}$  and each special vertical pair  $u, v$ .

We note, that in all cases  $\mathcal{A}_t(p, u, v)$  and  $\mathcal{B}_t(q, u, v)$  only depend on the labels of the nodes in the zone, for which  $(u, v)$  is special.

► **Lemma 3.6.** *From the stored values for functions  $\mathcal{A}_t$  and  $\mathcal{B}_t$  for special pairs, it is possible to compute  $\rho_t(v)$ , for arbitrary nodes  $v$ ,  $\mathcal{A}_t(p, u, u')$  for arbitrary pairs  $u \prec u'$  of siblings of  $t$  and  $\mathcal{B}_t(q, u, u')$  for arbitrary pairs  $u, u'$  of nodes, where  $u'$  is an ancestor of  $u$ , sequentially in constant time.*

This enables us to show the  $\mathcal{O}(n^\epsilon)$  work bound for label changes.

**Proof of Proposition 3.1.** To achieve the stated bound, we use the above algorithm with work parameter  $\theta = \frac{\epsilon}{2}$ . The algorithm uses a  $(\theta, 10)$ -bounded partition hierarchy, which exists thanks to Proposition 3.5.

As indicated before, the algorithm stores  $\mathcal{A}_t(\cdot, u, v)$  and  $\mathcal{B}_t(\cdot, u, v)$ , for all special pairs  $(u, v)$ . As already observed before, these values only depend on the labels of the nodes of the zone relative to which  $(u, v)$  is special. Therefore, if a node label is changed for some node  $x$ , values  $\mathcal{A}_t(\cdot, u, v)$  and  $\mathcal{B}_t(\cdot, u, v)$  need only be updated for special pairs of zones in

which  $x$  occurs. Since each node occurs in exactly  $h$  zones and each zone has  $\mathcal{O}(n^{2\theta}) = \mathcal{O}(n^\epsilon)$  special pairs,  $h \cdot \mathcal{O}(n^\epsilon)$  processors can be used, where every processor updates a single value in constant time and work, as is possible thanks to Lemma 3.2 and Lemma 3.6. Since the shape of the tree does not change we can assume a mapping from the updated node and the processor number to the special tuple that the respective processor recomputes. ◀

## 3.2 Structural Changes

In Proposition 3.1 only label changes were allowed, so the structure of the underlying tree did not change. In particular, there was no need to update any of the basic tree functions.

In this subsection, we consider structural changes of the tree. We show that the work bounds of Proposition 3.1 can still be met for the full data types  $\text{REGTREE}(L)$ .

► **Theorem 3.7.** *For each regular tree language  $L$  and each  $\epsilon > 0$ , there is a dynamic constant time parallel algorithm for  $\text{REGTREE}(L)$  that handles change operations with work  $\mathcal{O}(n^\epsilon)$  and answers query operations with constant work.*

In the next subsection, we describe the general strategy of the algorithm, define some notions that will be used and present its proof. Then, in a second subsection, we give some more detailed information about the data that is stored and how it can be maintained.

### 3.2.1 High-level description of the dynamic algorithm

Our approach generalises the algorithm of Subsection 3.1. It makes sure that, at any point in time, there is a valid partition hierarchy together with corresponding tree and automata functions. The general strategy of the dynamic algorithm is to add new leaves to their nearest zone. In principle, this is not hard to handle – unless it leads to a violation of a size constraint of some zone. As soon as zones exceed a certain size bound the affected parts of the hierarchy will thus be recomputed to ensure the size constraints.

For reasons that will become clearer below, we need to slightly modify the definition of partition hierarchies, basically by omitting the lowest two levels. To this end, we define 3-pruned partition hierarchies just like we defined partition hierarchies, but the lowest level is at height 3. More precisely, a *3-pruned partition hierarchy of height 3* is just a 3-zone, and *3-pruned partition hierarchies of height  $\ell > 3$*  are inductively defined just like partition hierarchies of height  $\ell$ . It is clear that a 3-pruned partition hierarchy exists for each tree by omitting the two lowest levels in the partition hierarchy computed in Proposition 3.5. Moreover, using a 3-pruned partition hierarchy as basis for our efficient label change approach still ensures the sequential constant time computation of arbitrary automaton function values from the stored values for special pairs. However, zones on the lowest level have size  $\mathcal{O}(n^{3\theta})$  leading to a work bound of  $\mathcal{O}(n^{6\theta})$  per change operation.

To ensure that at each point in time, a usable partition hierarchy is available, the general strategy is as follows: the algorithm starts from a *strong partition hierarchy* in which zones at level  $\ell$  have size at most  $\frac{1}{4}n^{\ell\theta}$ , well below the maximum allowed size of such a zone of  $n^{\ell\theta}$ . As soon as the size of a zone  $S$  at level  $\ell$  reaches its *warning limit*  $\frac{1}{2}n^{\ell\theta}$ , the algorithm starts to compute a new partition hierarchy for the parent zone  $S'$  of  $S$  at level  $\ell + 1$ . This computation is orchestrated in a way that makes sure that the new partition hierarchy for  $S'$  is ready (together with all required function values) before  $S$  reaches its size limit  $n^{\ell\theta}$ , at which point the old partition hierarchy for  $S'$  becomes useless.

Since a partition hierarchy of the whole tree together with the required function values has size  $\Omega(n)$ , its computation inherently requires that amount of work and it can probably not be done in constant time. Furthermore, since we aim at work  $\mathcal{O}(n^\epsilon)$  per operation, the

algorithm cannot afford to do the re-computation “as fast as possible” but rather needs to stretch over at least  $n^{1-\epsilon}$  steps. However, the fact that the tree can change during a re-computation poses a challenge: if many change operations happen with respect to a particular zone in a low level of the new partition hierarchy, this new zone might reach its warning limit and then its hard limit, before the overall re-computation of the hierarchy has finished. This challenge can be met by a careful orchestration of the re-computation.

We will next describe the data structure that the dynamic algorithm uses to orchestrate re-computations of partition hierarchies. As mentioned before, there will always be a valid partition hierarchy. However, for some zones, re-computations might be underway. The algorithm will always manage to complete the re-computation of a partition hierarchy for a zone of level  $\ell$ , before any of the subzones of level  $(\ell - 1)$  of the new partition reaches its warning limit. Therefore, for each zone within the data structure, there is always at most one partition hierarchy under construction, and therefore each zone has at any time at most two partition records. If a zone actually has two partition records, one of them contains a usable partition hierarchy. We formalise usability of a partition hierarchy by the term *operable* and tie the whole data structure together through the following notion of zone records. It is defined in an inductive fashion, together with the concept of partition records.

► **Definition 3.8.** A *zone record* of level 3 is a 3-zone. A *zone record* of level  $\ell > 3$  consists of an  $\ell$ -zone  $S$  and up to two partition records  $P_1, P_2$  of level  $\ell$  for  $S$ . If it has two partition records then  $P_1$  is complete and  $P_2$  is incomplete.

A *partition record*  $(Z, M)$  of level  $\ell > 3$  for an  $\ell$ -zone  $S$  consists of a set  $Z$  of zone records of level  $\ell - 1$  and a set  $M$  of zones, such that the zones from  $Z$  and the zones from  $M$  together constitute a partition of  $S$ . A partition record  $Z$  of level  $\ell$  is *valid*, if all zones of its zone records are actual  $(\ell - 1)$ -zones.

A zone record of level 3 is *operable*.

A partition record at level  $\ell > 3$  is *operable*, if it is valid and all its zone records are operable. A zone record of level  $\ell > 3$  is *operable*, if its first partition record is operable.

We refer to the hierarchical structure constituting the overall zone record as the *extended partition hierarchy*. Within the extended partition hierarchy, we are particularly interested in “operable substructures”. To this end, we associate with an operable zone record, the *primary partition hierarchy* that results from recursively picking the operable partition record from each zone record.

Altogether, the algorithm maintains an extended partition hierarchy for  $t$ .

Before we describe how the algorithm stores the extended partition hierarchy, we need two more concepts. For each zone record  $R$  of a level  $\ell$  there is a sequence  $R_h, \dots, R_\ell = R$  of zone records such that, for each  $i \geq \ell$ ,  $R_i$  is a zone record that occurs in a partition record of  $R_{i+1}$ . This sequence can be viewed as the *address* of  $R$  in the extended partition hierarchy. Furthermore, this address induces a *finger print* for  $R$ : the sequence  $\text{status}(R_h), \dots, \text{status}(R_\ell)$ , where  $\text{status}(R_i)$  is either *operable* or *in progress*. It is a simple but useful observation that if a tree node  $v$  occurs in two zones with zone records  $R \neq R'$  within the extended partition hierarchy, then the finger prints of  $R$  and  $R'$  are different. Consequently a tree node occurs in at most  $2^h$  and thus, a constant number of zones in the extended partition hierarchy.

Now we can describe, how the algorithm stores  $t$  and the extended partition hierarchy.

- A zone record of level 3 is represented as an array of  $\mathcal{O}(n^\theta)$  nodes.
- A zone record of a level  $\ell > 3$  consists of up to four boundary nodes and up to two pointers to partition records. The operable partition record is flagged.

- Each zone record of level  $\ell \geq 3$  with finger print  $pa$ , also stores a pointer to its zone on level  $\ell + 1$  with finger print  $p$ , and three pointers to the zone records of its parent, first child and right sibling zones.
- A partition record  $(Z, M)$  is represented as an array of zone records (some of which may be zones of  $M$ ). The zones records from  $Z$  are flagged.
- The nodes of  $t$  are stored in an array (in no particular order) together with pointers for the functions PARENT, LEFT-SIBLING, RIGHT-SIBLING, FIRST-CHILD, and LAST-CHILD.
- For each node  $v$ , and each possible finger print  $p$ , a pointer  $Z^p(v)$  to its zone with finger print  $p$ .

Now we are prepared to outline the proof of Theorem 3.7.

**Proof (of Theorem 3.7).** Let  $\mathcal{B}$  be a DTA for the regular tree language  $L$  and let  $\theta = \frac{\varepsilon}{7}$ . The dynamic algorithm stores  $t$  and an extended partition hierarchy as described above. It also stores some additional function values, including values for the automata functions, that will be specified in Subsubsection 3.2.2.

Some functions are independent from zones and are stored for all nodes. Some other functions are independent from zones but are only stored for particular node tuples that are induced from zones (like it was already the case for the automata functions in Subsection 3.1) and some functions are actually defined for (tuples of) zones.

After each change operation, the algorithm updates function values, pursues re-computations of hierarchies and computes function values that are needed for newly established zones. It starts a re-computation for a zone  $S$ , whenever one of its subzones reaches its warning limit. It starts a re-computation of the overall zone, whenever the number of nodes of  $t$  reaches  $\frac{1}{2}n$ .

The algorithm has one thread for each zone with an ongoing re-computation, that is, for each zone whose zone record is not yet operable.

A re-computation for a zone at level  $\ell$  requires the computation of  $\mathcal{O}(n^\theta)$  zones of level  $\ell - 1$ , each of which yields re-computations of  $\mathcal{O}(n^\theta)$  zones of level  $\ell - 2$  and so forth, down to level 3. It is easy to see that the overall number of zones that needs to be computed during a re-computation of a zone at level  $\ell$  is bounded by  $\mathcal{O}(n^{(\ell-3)\theta})$ . The re-computation of the overall zone requires the computation of at most  $\mathcal{O}(n^{1-3\theta})$  zones. We show in Lemma 3.9 that, in the presence of a primary partition hierarchy for the overall zone, the computation of a new zone is possible in constant time with work  $\mathcal{O}(n^{6\theta})$ .

The thread for the re-computation of a zone at level  $\ell$  thus (first) consists of  $\mathcal{O}(n^{(\ell-3)\theta})$  computations of component zones, each of which is carried out in constant time with work  $\mathcal{O}(n^{6\theta})$ . We refer to such a re-computation as a round. A thread thus consists of  $\mathcal{O}(n^{(\ell-3)\theta})$  rounds of zone computations. The thread follows a breadth-first strategy, that is, it first computes all zones of level  $\ell - 1$  then the sub-zones of those zones at level  $\ell - 2$  and so forth. Once the zone record of a zone  $S$  is operable, the thread computes in its second phase all function values associated to  $S$ . This can be done in constant time with work  $\mathcal{O}(n^{7\theta})$  per sub-zone of  $S$ , as is shown in the full version of this paper. That is, it requires at most  $\mathcal{O}(n^{(\ell-3)\theta})$  additional rounds.

We note that it does not matter if the primary partition hierarchy  $H$  required for Lemma 3.9 changes during the computation of a thread, since  $H$  is only used to make the identification of a new zone more efficient.

To address the above mentioned challenge, the algorithm starts a separate thread for each zone that is newly created during this process. That is, for each zone at level  $\ell - 1$ , an additional re-computation thread is started, as soon as the zone is created.

Now we can state the orchestration strategy for re-computations. This strategy is actually very simple:

**Re-computation strategy:** After each change operation affecting some node  $v$ , the algorithm performs one computation round, for all threads of zones  $S$ , at any level, with  $v \in S$ .

That is, thanks to the above observation, after a change operation, there are at most  $2^h$  threads for which one computation round is performed. Since  $2^h$  is a constant, these computations together require work at most  $\mathcal{O}(n^{7\theta})$ .

On the other hand, the whole re-computation for a zone  $S$  at level  $\ell$ , including the computation of the relevant function values, is finished after at most  $\mathcal{O}(n^{(\ell-3)\theta})$  change operations that affect  $S$ . Since  $\frac{1}{2}n^{(\ell-1)\theta}$  leaf additions are needed to let a sub-zone  $S'$  grow from the warning limit  $\frac{1}{2}n^{(\ell-1)\theta}$  to the hard limit  $n^{(\ell-1)\theta}$ , it is guaranteed that the re-computation thread for  $S$  is completed, before  $S'$  grows too large. In fact, this is exactly, why partition hierarchies are 3-pruned. When a re-computation of the overall zone was triggered by the size of  $t$ ,  $n$  is doubled as soon as this re-computation is completed.

Thanks to Lemma 3.11 the overall work to update the stored function values for all affected zones (in constant time) after a change operation is  $\mathcal{O}(n^{3\theta})$ .

Altogether, the statement of the theorem follows by choosing  $\theta = \frac{\epsilon}{7}$ . ◀

We state the lemma about the computation of new zones next. The partition hierarchy is used as a means to assign evenly distributed nodes to processors and to do parallel search for nodes with a particular property regarding the number of their descendants.

► **Lemma 3.9.** *Given a tree  $t$ , a 3-pruned partition hierarchy  $H$  of  $t$ , and a zone  $S$  with at least  $m$  nodes,  $S$  can be partitioned into at most five zones, one of which has at least  $\frac{1}{2}m$  and at most  $m$  nodes, in constant time with work  $\mathcal{O}(n^{6\theta})$ .*

### 3.2.2 Maintaining functions

In Subsection 3.1, the tree functions were static and given by the initialisation. Only the automata functions needed to be updated. However, if leaf insertions are allowed, the tree functions can change. To keep the algorithm efficient, the special pairs need to be adapted to the evolution of the partition hierarchy, and tree functions can no longer be stored for all possible arguments. Furthermore, additional tree functions and functions defined for zones will be used.

The stored information suffices to compute all required functions in constant time, and almost all of them with constant work.

► **Lemma 3.10.** *Given a tree  $t$ , a 3-pruned partition hierarchy  $H$  of  $t$ , and the stored information as described above, for each  $\theta > 0$ , the `CHILD` function can be evaluated in  $\mathcal{O}(1)$  time with work  $\mathcal{O}(n^\theta)$ . All other functions can be evaluated for all tuples with constant work.*

Furthermore, all stored information can be efficiently updated, with the help of and in accordance with the current primary partition hierarchy.

► **Lemma 3.11.** *Let  $\theta > 0$  and  $H$  be a 3-pruned partition hierarchy of  $t$  with automata and tree functions. The stored information described above can be maintained after each `Relabel` and `AddChild` operation in constant time with  $\mathcal{O}(n^{6\theta})$  work per operation.*

## 4 Maintaining context-free languages

As mentioned in the introduction, an analysis of the dynamic program that was used in [7] to show that context-free languages can be maintained in DynFO yields the following result.

► **Theorem 4.1** ([7, Proposition 5.3]). *For each context-free language  $L$ , there is a dynamic constant-time parallel algorithm on a CRCW PRAM for  $\text{CFL}(L)$  with  $\mathcal{O}(n^7)$  work.*

There is a huge gap between this upper bound and the conditional lower bound of  $\mathcal{O}(n^{\omega-1-\epsilon})$ , for any  $\epsilon > 0$ , derived from the  $k$ -Clique conjecture [1], where  $\omega < 2.373$  [11]. Our attempts to make this gap significantly smaller, have not been successful yet. However, for realtime deterministic context-free languages and visibly pushdown languages, more efficient dynamic algorithms are possible, as shown in the following two subsections.

### 4.1 Deterministic context-free languages

Realtime deterministic context-free languages are decided by deterministic PDAs without  $\lambda$ -transitions (RDPDAs).

► **Theorem 4.2.** *For each realtime deterministic context-free language  $L$  and each  $\epsilon > 0$ , there is a dynamic constant-time parallel algorithm on a CRCW PRAM for  $\text{CFL}(L)$  with  $\mathcal{O}(n^{3+\epsilon})$  work.*

Given an RDPDA  $\mathcal{A}$  for  $L$ , a configuration  $C = (p, u, s)$  consists of a state  $p$ , a string  $u$  that is supposed to be read by  $\mathcal{A}$  and a string  $s$ , the initial stack content. We use the following functions  $\delta_{\text{state}}$ ,  $\delta_{\text{stack}}$ , and  $\delta_{\text{empty}}$  to describe the behaviour of  $\mathcal{A}$  on configurations.

- $\delta_{\text{state}}(C)$  yields the last state of  $\text{run}(C)$ .
- $\delta_{\text{stack}}(C)$  yields the stack content at the end of  $\text{run}(C)$ .
- $\delta_{\text{empty}}(C)$  is the position in  $u$ , after which  $\text{run}(C)$  empties its stack. It is zero, if this does not happen at all.

The algorithm maintains the following information, for each simple configuration  $C = (p, u, \tau)$ , where  $u = w[i, j]$ , for some  $i \leq j$ , for each suffix  $v = w[m, n]$  of  $w$ , where  $j < m$ , each state  $q$ , and some  $k \leq n$ .

- $\hat{\delta}(C)$  defined as the tuple  $(\delta_{\text{state}}(C), |\delta_{\text{stack}}(C)|, \text{top}_1(\delta_{\text{stack}}(C)), \delta_{\text{empty}}(C))$ , consisting of the state, the height of the stack, the top symbol of the stack, at the end of the run on  $C$  and the position where the run ends. If the run empties the stack prematurely or at the end of  $u$ , then  $\text{top}_1(\delta_{\text{stack}}(C))$  is undefined;
- $\text{push-pos}(C, k)$ , defined as the length of the longest prefix  $x$  of  $u$ , such that  $|\delta_{\text{stack}}(p, x, \tau)| = k$ . Informally this is the position of  $u$  at which the  $k$ -th symbol of  $\delta_{\text{stack}}(C)$ , counted from the bottom, is written;
- $\text{pop-pos}(C, q, v, k)$ , defined as the pair  $(o, r)$ , where  $o$  is the length of the prefix  $v'$  of  $v$ , for which  $\text{run}(q, v, \text{top}_k(\delta_{\text{stack}}(C)))$  empties its stack at the last symbol of  $v'$ , and  $r$  is the state it enters.

However, tuples for push-pos and pop-pos are only stored for values  $k$  of the form  $an^{b\theta}$ , for integers  $b < \frac{1}{\theta}$  and  $a \leq n^\theta$ , for some fixed  $\theta > 0$ . A more detailed account is given in the full version of this paper.

### 4.2 Visibly pushdown languages

Visibly pushdown languages are a subclass of realtime deterministic CFLs. They use *pushdown alphabets* of the form  $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_{\text{int}})$  and deterministic PDA that always push a symbol when reading a symbol from  $\Sigma_c$ , pop a symbol when reading a symbol from  $\Sigma_r$  and leave the stack unchanged otherwise. We refer to [2] for more information.

There is a correspondence between wellformed strings over a pushdown alphabet and labelled trees, where each matching pair  $(a, b)$  of a call symbol from  $\Sigma_c$  and a return symbol from  $\Sigma_r$  is represented by an inner node with label  $(a, b)$  and each other symbol by a leaf. From Theorem 3.7 and this correspondence the following can be concluded.

► **Proposition 4.3.** *For each visibly pushdown language  $L$  and each  $\epsilon > 0$ , there is a dynamic constant-time parallel algorithm on a CRCW PRAM for  $VPL^-(L)$  with  $\mathcal{O}(n^\epsilon)$  work.*

Here,  $VPL^-(L)$  only allows the following change operations:

- Replacement of a symbol by a symbol of the same type;
- Insertion of an internal symbol from  $\Sigma_{\text{int}}$  before a return symbol;
- Replacement of an internal symbol by two symbols  $ab$ , where  $a \in \Sigma_c$  and  $b \in \Sigma_r$ .

For arbitrary symbol replacements and insertions, there is a much less work-efficient algorithm which, however, is still considerably more efficient than the algorithm for DCFLs.

► **Theorem 4.4.** *For each visibly pushdown language  $L$  and each  $\epsilon > 0$ , there is a dynamic constant-time parallel algorithm on a CRCW PRAM for  $VPL(L)$  with  $\mathcal{O}(n^{2+\epsilon})$  work.*

The work improvement mainly relies on the fact that how the height of the stack evolves during a computation only depends on the types of symbols.

## 5 Conclusion

We have shown that the good work bounds for regular string languages from [11] carry over to regular tree languages, even under some structural changes of the tree. In turn they also hold for visibly pushdown languages under limited change operations. For realtime deterministic context-free languages and visibly pushdown languages under more general change operations better work bounds than for context-free languages could be shown.

There are plenty of questions for further research, including the following: are there other relevant change operations for trees that can be handled with work  $\mathcal{O}(n^\epsilon)$ ? What are good bounds for further operations? Can the bounds for context-free languages be improved? Can the  $\mathcal{O}(n^{3+\epsilon})$  be shown for arbitrary (not necessarily realtime) DCFLs? And the most challenging: are there further lower bound results that complement our upper bounds?

---

## References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–211, 2004. doi:10.1145/1007352.1007390.
- 3 Antoine Amarilli, Pierre Bourhis, and Stefan Mengel. Enumeration on trees under relabelings. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory (ICDT)*, volume 98 of *LIPICs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICDT.2018.5.
- 4 Mikołaj Bojańczyk. Algorithms for regular languages that use algebra. *SIGMOD Rec.*, 41(2):5–14, 2012. doi:10.1145/2350036.2350038.
- 5 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. hal-03367725, 2008.
- 6 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

- 7 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012. doi:10.1145/2287718.2287719.
- 8 Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.
- 9 Joseph F. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
- 10 Sushant Patnaik and Neil Immerman. Dyn-FO: A Parallel, Dynamic Complexity Class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.
- 11 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures – 24th International Conference (FOSSACS)*, pages 490–509, 2021. doi:10.1007/978-3-030-71995-1\_25.
- 12 Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of regular languages: Big changes, small work. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL)*, pages 35:1–35:19, 2023. doi:10.4230/LIPIcs.CSL.2023.35.
- 13 Peter van Emde Boas. Machine models and simulation. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier and MIT Press, 1990.
- 14 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 887–898, 2012. doi:10.1145/2213977.2214056.



# Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors

Tim Seppelt  

RWTH Aachen University, Germany

---

## Abstract

Two graphs  $G$  and  $H$  are *homomorphism indistinguishable* over a class of graphs  $\mathcal{F}$  if for all graphs  $F \in \mathcal{F}$  the number of homomorphisms from  $F$  to  $G$  is equal to the number of homomorphisms from  $F$  to  $H$ . Many natural equivalence relations comparing graphs such as (quantum) isomorphism, spectral, and logical equivalences can be characterised as homomorphism indistinguishability relations over certain graph classes.

Abstracting from the wealth of such instances, we show in this paper that equivalences w.r.t. any *self-complementarity* logic admitting a characterisation as homomorphism indistinguishability relation can be characterised by homomorphism indistinguishability over a minor-closed graph class. Self-complementarity is a mild property satisfied by most well-studied logics. This result follows from a correspondence between closure properties of a graph class and preservation properties of its homomorphism indistinguishability relation.

Furthermore, we classify all graph classes which are in a sense finite (*essentially profinite*) and satisfy the maximality condition of being *homomorphism distinguishing closed*, i.e. adding any graph to the class strictly refines its homomorphism indistinguishability relation. Thereby, we answer various questions raised by Roberson (2022) on general properties of the homomorphism distinguishing closure.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory

**Keywords and phrases** homomorphism indistinguishability, graph minor, logic

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.82

**Related Version** *Full Version*: <https://arxiv.org/abs/2302.11290> [35]

**Funding** *Tim Seppelt*: German Research Council (DFG) within Research Training Group 2236 (UnRAVeL).

**Acknowledgements** I would like to thank David E. Roberson, Louis Härtel, Martin Grohe, Gaurav Rattan, and Christoph Standke for fruitful discussions.

## 1 Introduction

In 1967, Lovász [23] proved that two graphs  $G$  and  $H$  are isomorphic if and only if they are *homomorphism indistinguishable* over all graphs, i.e. for every graph  $F$ , the number of homomorphisms from  $F$  to  $G$  is equal to the number of homomorphisms from  $F$  to  $H$ . Since then, homomorphism indistinguishability over restricted graph classes has emerged as a powerful framework for capturing a wide range of equivalence relations comparing graphs. For example, two graphs have cospectral adjacency matrices iff they are homomorphism indistinguishable over all cycles, cf. [13]. They are quantum isomorphic iff they are homomorphism indistinguishable over all planar graphs [25].

Most notably, equivalences with respect to many logic fragments can be characterised as homomorphism indistinguishability relations over certain graph classes [19, 12, 27, 31]. For example, two graphs satisfy the same sentences of  $k$ -variable counting logic iff they are homomorphism indistinguishable over graphs of treewidth less than  $k$  [14]. All graph classes



© Tim Seppelt;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 82; pp. 82:1–82:15

Leibniz International Proceedings in Informatics



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

■ **Table 1** Overview of results on equivalent properties of a homomorphism distinguishing closed graph class  $\mathcal{F}$  and of its homomorphism indistinguishability relation  $\equiv_{\mathcal{F}}$ .

Closure property of $\mathcal{F}$	Preservation property of $\equiv_{\mathcal{F}}$	Theorem
taking minors	complements	Theorem 9
taking summands	disjoint unions	Theorem 5
taking subgraphs	full complements	Theorem 6
taking induced subgraphs	left lexicographic products	Theorem 12
contracting edges	right lexicographic products	Theorem 13

featured in such characterisations are minor-closed and hence of a particularly enjoyable structure. The main result of this paper asserts that this is not a mere coincidence: In fact, logical equivalences and homomorphism indistinguishability over minor-closed graph classes are intimately related.

To make this statement precise, the term “logic” has to be formalised. Following [15], a *logic on graphs* is a pair  $(\mathbb{L}, \models)$  of a class  $\mathbb{L}$  of *sentences* and an isomorphism-invariant<sup>1</sup> *model relation*  $\models$  between graphs and sentences. Two graphs  $G$  and  $H$  are  *$\mathbb{L}$ -equivalent* if  $G \models \varphi$  iff  $H \models \varphi$  for all  $\varphi \in \mathbb{L}$ . One may think of a logic on graphs as a collection of isomorphism-invariant graph properties. A logic is called *self-complementary* if for every  $\varphi \in \mathbb{L}$  there is an element  $\bar{\varphi} \in \mathbb{L}$  such that  $G \models \varphi$  if and only if  $\bar{G} \models \bar{\varphi}$ . Here,  $\bar{G}$  denotes the complement graph of  $G$ . Roughly speaking, a fragment/extension  $\mathbb{L}$  of first-order logic is self-complementary if expressions of the form  $Exy$  can be replaced by  $\neg Exy \wedge (x \neq y)$  in every formula while remaining in  $\mathbb{L}$ . This lax requirement is satisfied by many logics including first-order logic, counting logic, second-order logic, fixed-point logics, and bounded variable, quantifier depth, or quantifier prefix fragments of these. All these examples are subject to the following result:

► **Theorem 1.** *Let  $(\mathbb{L}, \models)$  be a self-complementary logic on graphs for which there exists a graph class  $\mathcal{F}$  such that two graphs  $G$  and  $H$  are homomorphism indistinguishable over  $\mathcal{F}$  if and only if they are  $\mathbb{L}$ -equivalent. Then there exists a minor-closed graph class  $\mathcal{F}'$  whose homomorphism indistinguishability relation coincides with  $\mathbb{L}$ -equivalence.*

Theorem 1 can be used to rule out that a given logic has a homomorphism indistinguishability characterisation (Corollary 17). Furthermore, it allows to use the deep results of graph minor theory to study the expressive power of logics on graphs (Theorem 22).

Theorem 1 is product of a more fundamental study of the properties of homomorphism indistinguishability relations. In several instances, it is shown that closure properties of a graph class  $\mathcal{F}$  correspond to preservation properties of its homomorphism indistinguishability relation  $\equiv_{\mathcal{F}}$ . These efforts yield answers to several open questions from [32]. A prototypical result is Theorem 2, from which Theorem 1 follows. For further results in the same vein concerning other closure properties, e.g. under taking subgraphs, see Table 1.

► **Theorem 2.** *Let  $\mathcal{F}$  be a homomorphism distinguishing closed graph class. Then  $\mathcal{F}$  is minor-closed if and only if  $\equiv_{\mathcal{F}}$  is preserved under taking complements, i.e. for all simple graphs  $G$  and  $H$  it holds that  $G \equiv_{\mathcal{F}} H$  if and only if  $\bar{G} \equiv_{\mathcal{F}} \bar{H}$ .*

Here, the graph class  $\mathcal{F}$  is *homomorphism distinguishing closed* [32] if for every  $K \notin \mathcal{F}$  there exist graphs  $G$  and  $H$  which are homomorphism indistinguishable over  $\mathcal{F}$  but differ in the number of homomorphisms from  $K$ . In other words, adding even a single graph to  $\mathcal{F}$  would change its homomorphism indistinguishability relation.

<sup>1</sup> For all  $\varphi \in \mathbb{L}$  and graphs  $G$  and  $H$  such that  $G \cong H$ , it holds that  $G \models \varphi$  iff  $H \models \varphi$ .

Proving that a graph class is homomorphism distinguishing closed is a pathway to separating equivalence relations comparing graphs [33]. However, establishing this property is a notoriously hard task. Thus, a general result establishing the homomorphism distinguishing closedness of a wide range of graph classes would be desirable. In [32], Roberson conjectured that *every graph class closed under taking minors and disjoint unions is homomorphism distinguishing closed*. At present, this conjecture has only been verified for few graph classes [32, 28].

Our final result confirms Roberson’s conjecture for all graph classes which are in a certain sense finite. The expressive power of homomorphism counts from finitely many graphs is of particular importance in practice. Applications include the design of graph kernels [21], motif counting [2, 26], or machine learning on graphs [6, 30, 20]. A theoretical interest stems for example from database theory where homomorphism counts correspond to results of queries under bag-semantics [8, 22], see also [9].

Since every homomorphism distinguishing closed graph class is closed under taking disjoint unions, infinite graph classes arise inevitably when studying homomorphism indistinguishability over finite graph classes. We introduce the notions of *essentially finite* and *essentially profinite* graph classes (Definition 23) in order to capture the nevertheless limited behaviour of graph classes arising from the finite. Examples for essentially profinite graph classes include the class of all minors of a fixed graph and the class of cluster graphs, i.e. disjoint unions of arbitrarily large cliques. In Theorem 24, the essentially profinite graph classes which are homomorphism distinguishing closed are fully classified. Thereby, the realm of available examples of homomorphism distinguishing closed graph classes is drastically enlarged. This classification has the following readily-stated corollary:

► **Theorem 3.** *Every essentially profinite union-closed graph class  $\mathcal{F}$  which is closed under taking summands<sup>2</sup> is homomorphism distinguishing closed. In particular, Roberson’s conjecture holds for all essentially profinite graph classes.*

## 2 Preliminaries

All graphs in this article are finite, undirected, and without multiple edges. A *simple graph* is a graph without loops. A *homomorphism* from a graph  $F$  to a graph  $G$  is a map  $h: V(F) \rightarrow V(G)$  such that  $h(u)h(v) \in E(G)$  whenever  $uv \in E(F)$  and vertices carrying loops are mapped to vertices carrying loops. Write  $\text{hom}(F, G)$  for the number of homomorphisms from  $F$  to  $G$ . For a class of graphs  $\mathcal{F}$  and graphs  $G$  and  $H$ , write  $G \equiv_{\mathcal{F}} H$  if  $\text{hom}(F, G) = \text{hom}(F, H)$  for all  $F \in \mathcal{F}$ , i.e.  $G$  and  $H$  are *homomorphism indistinguishable over  $\mathcal{F}$* . With the exception of Section 3.2, the graphs in  $\mathcal{F}$ ,  $G$  and  $H$  will be simple. Following [32], the *homomorphism distinguishing closure* of  $\mathcal{F}$  is

$$\text{cl}(\mathcal{F}) := \{K \text{ simple graph} \mid \forall \text{ simple graphs } G, H. \quad G \equiv_{\mathcal{F}} H \Rightarrow \text{hom}(K, G) = \text{hom}(K, H)\}.$$

Intuitively,  $\text{cl}(\mathcal{F})$  is the ‘largest’ graph class whose homomorphism indistinguishability relation coincides with the one of  $\mathcal{F}$ . A graph class  $\mathcal{F}$  is *homomorphism distinguishing closed* if  $\text{cl}(\mathcal{F}) = \mathcal{F}$ . Note that  $\text{cl}$  is a closure operator in the sense that  $\text{cl}(\mathcal{F}) \subseteq \text{cl}(\mathcal{F}')$  if  $\mathcal{F} \subseteq \mathcal{F}'$  and  $\text{cl}(\text{cl}(\mathcal{F})) = \text{cl}(\mathcal{F})$  for all graph classes  $\mathcal{F}$  and  $\mathcal{F}'$ .

For graphs  $G$  and  $H$ , write  $G + H$  for their *disjoint union* and  $G \times H$  for their *categorical product*, i.e.  $V(G \times H) := V(G) \times V(H)$  and  $gh$  and  $g'h'$  are adjacent in  $G \times H$  iff  $gg' \in E(G)$  and  $hh' \in E(H)$ . The *lexicographic product*  $G \cdot H$  is defined as the graph with vertex set

<sup>2</sup> A graph class  $\mathcal{F}$  is *closed under taking summands* if for all  $F \in \mathcal{F}$  which is the disjoint union of two graphs  $F_1 + F_2 = F$  also  $F_1, F_2 \in \mathcal{F}$ .

$V(G) \times V(H)$  and edges between  $gh$  and  $g'h'$  iff  $g = g'$  and  $hh' \in E(H)$  or  $gg' \in E(G)$ . It is well-known, cf. e.g. [24, (5.28)–(5.30)], that for all graphs  $F_1, F_2, G_1, G_2$ , and all connected graphs  $K$ ,

$$\text{hom}(F_1 + F_2, G) = \text{hom}(F_1, G) \text{hom}(F_2, G), \quad (1)$$

$$\text{hom}(F, G_1 \times G_2) = \text{hom}(F, G_1) \text{hom}(F, G_2), \text{ and} \quad (2)$$

$$\text{hom}(K, G_1 + G_2) = \text{hom}(K, G_1) + \text{hom}(K, G_2). \quad (3)$$

The *complement* of a simple graph  $F$  is the simple graph  $\bar{F}$  with  $V(\bar{F}) = V(F)$  and  $E(\bar{F}) = \binom{V(F)}{2} \setminus E(F)$ . Observe that  $\bar{\bar{F}} = F$  for all simple graphs  $F$ . The *full complement* of a graph  $G$  is the graph  $\widehat{G}$  obtained from  $G$  by replacing every edge with a non-edge and every loop with a non-loop, and vice-versa.

The *quotient*  $F/\mathcal{P}$  of a simple graph  $F$  by a partition  $\mathcal{P}$  of  $V(F)$  is the simple graph with vertex set  $\mathcal{P}$  and edges  $PQ$  for  $P \neq Q$  iff there exist vertices  $p \in P$  and  $q \in Q$  such that  $pq \in E(F)$ . For a set  $X$ , write  $\Pi(X)$  for the set of all partitions of  $X$ . We do not include loops in order to state Theorem 14 succinctly.

A graph  $F'$  can be obtained from a simple graph  $F$  by *contracting edges* if there is a partition  $\mathcal{P} \in \Pi(V(F))$  such that  $F[P]$  is connected for all  $P \in \mathcal{P}$  and  $F' \cong F/\mathcal{P}$ .

For a graph  $F$  and a set  $P \subseteq V(F)$ , write  $F[P]$  for the *subgraph induced by  $P$* , i.e. the graph with vertex set  $P$  and edges  $uv$  if  $u, v \in P$  and  $uv \in E(F)$ . A graph  $F'$  is a *subgraph* of  $F$ , in symbols  $F' \subseteq F$  if  $V(F') \subseteq V(F)$  and  $E(F') \subseteq E(F)$ . A *minor* of a simple graph  $F$  is a subgraph of a graph which can be obtained from  $F$  by contracting edges.

### 3 Closure Properties Correspond to Preservation Properties

This section is concerned with the interplay of closure properties of a graph class  $\mathcal{F}$  and preservation properties of its homomorphism indistinguishability relation  $\equiv_{\mathcal{F}}$ . The central results of this section are Theorem 2 and the other results listed in Table 1.

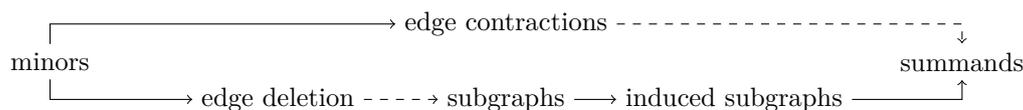
The relevance of the results is twofold: On the one hand, they yield that if a graph class  $\mathcal{F}$  has a certain closure property then so does  $\text{cl}(\mathcal{F})$ . In the case of minor-closed graph families, this provides evidence for Robertson's conjecture [32]. On the other hand, they establish that equivalence relations comparing graphs which are preserved under certain operations coincide with the homomorphism indistinguishability relation over a graph class with a certain closure property, if they are homomorphism indistinguishability relations at all. Further consequences are discussed in Sections 3.5 and 4. Essential to all proofs is the following lemma:

► **Lemma 4.** *Let  $\mathcal{F}$  and  $\mathcal{L}$  be classes of simple graphs. Suppose  $\mathcal{L}$  is finite and that its elements are pairwise non-isomorphic. Let  $\alpha: \mathcal{L} \rightarrow \mathbb{R} \setminus \{0\}$ . If for all simple graphs  $G$  and  $H$*

$$G \equiv_{\mathcal{F}} H \implies \sum_{L \in \mathcal{L}} \alpha_L \text{hom}(L, G) = \sum_{L \in \mathcal{L}} \alpha_L \text{hom}(L, H)$$

*then  $\mathcal{L} \subseteq \text{cl}(\mathcal{F})$ .*

**Proof.** The following argument is due to [11, Lemma 3.6]. Let  $n$  be an upper bound on the number of vertices of graphs in  $\mathcal{L}$  and let  $\mathcal{L}'$  denote the class of all graphs on at most  $n$  vertices. By classical arguments [23], the matrix  $M := (\text{hom}(K, L))_{K, L \in \mathcal{L}'}$  is invertible. Extend  $\alpha$  to a function  $\alpha': \mathcal{L}' \rightarrow \mathbb{R}$  by setting  $\alpha'(L) := \alpha(L)$  for all  $L \in \mathcal{L}$  and  $\alpha'(L') := 0$  for all  $L' \in \mathcal{L}' \setminus \mathcal{L}$ . If  $G \equiv_{\mathcal{F}} H$  then  $G \times K \equiv_{\mathcal{F}} H \times K$  for all graphs  $K$ . Hence, by Equation (2),



■ **Figure 1** Relationship between closure properties of homomorphism distinguishing closed graph classes. The non-obvious implications (dashed) are proven in Lemmas 8 and 16.

$$\sum_{L \in \mathcal{L}'} \text{hom}(L, K) \alpha_L \text{hom}(L, G) = \sum_{L \in \mathcal{L}'} \text{hom}(L, K) \alpha_L \text{hom}(L, H).$$

Both sides can be read as the product of the matrix  $M^T$  with a vector of the form  $(\alpha_L \text{hom}(L, -))_{L \in \mathcal{L}'}$ . By multiplying from the left with the inverse of  $M^T$ , it follows that  $\alpha_L \text{hom}(L, G) = \alpha_L \text{hom}(L, H)$  for all  $L \in \mathcal{L}'$  which in turn implies that  $\text{hom}(L, G) = \text{hom}(L, H)$  for all  $L \in \mathcal{L}$ . Thus,  $\mathcal{L} \subseteq \text{cl}(\mathcal{F})$ . ◀

In the setting of Lemma 4, we say that the relation  $\equiv_{\mathcal{F}}$  determines the linear combination  $\sum_{L \in \mathcal{L}} \alpha_L \text{hom}(L, -)$ . Note that it is essential for the argument to carry through that the elements of  $\mathcal{L}$  are pairwise non-isomorphic and that  $\alpha_L \neq 0$  for all  $L$ . Efforts will be undertaken to establish this property for certain linear combinations in the subsequent sections.

### 3.1 Taking Summands and Preservation under Disjoint Unions

In this section, the strategy yielding the results in Table 1 is presented for the rather simple case of Theorem 5. This theorem relates the property of a graph class  $\mathcal{F}$  to be closed under taking summands to the property of  $\equiv_{\mathcal{F}}$  to be preserved under disjoint unions. This closure property is often assumed in the context of homomorphism indistinguishability [1, 32] and fairly mild. It is the most general property among those studied here, cf. Figure 1. Theorem 5 answers a question from [32, p. 7] affirmatively: Is it true that if  $\equiv_{\mathcal{F}}$  is preserved under disjoint unions then  $\text{cl}(\mathcal{F})$  is closed under taking summands?

► **Theorem 5.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under taking summands, i.e. if  $F_1 + F_2 \in \mathcal{F}$  then  $F_1, F_2 \in \mathcal{F}$ ,
- (ii)  $\equiv_{\mathcal{F}}$  is preserved under disjoint unions, i.e. for all simple graphs  $G, G', H,$  and  $H'$ , if  $G \equiv_{\mathcal{F}} G'$  and  $H \equiv_{\mathcal{F}} H'$  then  $G + H \equiv_{\mathcal{F}} G' + H'$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is closed under taking summands.

the implications  $i \Rightarrow ii \Leftrightarrow iii$  hold.

**Proof.** The central idea is to write, given graphs  $F, G,$  and  $H,$  the quantity  $\text{hom}(F, G + H)$  as expression in  $\text{hom}(F', G)$  and  $\text{hom}(F', H)$  where the  $F'$  range over summands of  $F$ . To this end, write  $F = C_1 + \dots + C_r$  as disjoint union of its connected components. Then,

$$\begin{aligned} \text{hom}(F, G + H) &\stackrel{(1)}{=} \prod_{i=1}^r \text{hom}(C_i, G + H) \stackrel{(3)}{=} \prod_{i=1}^r (\text{hom}(C_i, G) + \text{hom}(C_i, H)) \\ &\stackrel{(1)}{=} \sum_{I \subseteq [r]} \text{hom}\left(\sum_{i \in I} C_i, G\right) \text{hom}\left(\sum_{i \in [r] \setminus I} C_i, H\right). \end{aligned} \tag{4}$$

In particular, if  $\mathcal{F}$  is closed under taking summands then  $\sum_{i \in I} C_i \in \mathcal{F}$  for all  $I \subseteq [r]$ . Thus, i implies ii.

Assume ii and let  $F \in \text{cl}(\mathcal{F})$ . Write as above  $F = C_1 + \cdots + C_r$  as disjoint union of its connected components. By the assumption that  $\equiv_{\mathcal{F}}$  is preserved under disjoint unions, for all graphs  $G$  and  $G'$ , if  $G \equiv_{\mathcal{F}} G'$  then  $G + F \equiv_{\mathcal{F}} G' + F$  and hence  $\text{hom}(F, G + F) = \text{hom}(F, G' + F)$ . By Equation (4) with  $H = F$ , the relation  $\equiv_{\mathcal{F}}$  determines the linear combination  $\sum_{I \subseteq [r]} \text{hom}(\sum_{i \in I} C_i, -) \text{hom}(\sum_{i \in [r] \setminus I} C_i, F)$ . Note that it might be the case that  $\sum_{i \in I} C_i \cong \sum_{j \in J} C_j$  for some  $I \neq J$ . Grouping such summands together and adding their coefficients yields a linear combination satisfying the assumptions of Lemma 4 since  $\text{hom}(C_i, F) > 0$  for all  $i \in [r]$ . Hence,  $\sum_{i \in I} C_i \in \text{cl}(\mathcal{F})$  for all  $I \subseteq [r]$  and iii follows.

The implication iii  $\Rightarrow$  ii follows from i  $\Rightarrow$  ii for  $\text{cl}(\mathcal{F})$  since  $\equiv_{\mathcal{F}}$  and  $\equiv_{\text{cl}(\mathcal{F})}$  coincide.  $\blacktriangleleft$

The proofs of the other results in Table 1 are conceptually similar to the just completed proof. The general idea can be briefly described as follows:

1. Derive a linear expression similar to Equation (4) for the number of homomorphisms from  $F$  into the graph constructed using the assumed preservation property of  $\equiv_{\mathcal{F}}$ , e.g. the graph  $G + H$  in the case of Theorem 5. These linear combinations typically involve sums over subsets  $U$  of vertices or edges of  $F$ , each contributing a summand of the form  $\alpha_U \text{hom}(F_U, -)$  where  $\alpha_U$  is some coefficient and  $F_U$  is a graph constructed from  $F$  using  $U$ . Hence, if  $\mathcal{F}$  is closed under the construction transforming  $F$  to  $F_U$  then  $\equiv_{\mathcal{F}}$  has the desired preservation property.
2. In general, it can be that  $F_U$  and  $F_{U'}$  are isomorphic even though  $U \neq U'$ , e.g. in Equation (4) if  $F$  contains two isomorphic connected components. In order to apply Lemma 4, one must group the summands  $\alpha_U \text{hom}(F_U, -)$  by the isomorphism type  $F'$  of the  $F_U$ . The coefficient of  $\text{hom}(F', -)$  in the new linear combination ranging over pairwise non-isomorphic graphs is the sum of  $\alpha_U$  over all  $U$  such that  $F_U \cong F'$ . Once it is established that this coefficient is non-zero, it follows that if  $\equiv_{\mathcal{F}}$  has the preservation property then  $\text{cl}(\mathcal{F})$  has the desired closure property.

### 3.2 Taking Subgraphs and Preservation under Full Complements

The strategy which yielded Theorem 5 can be extended to obtain the following Theorem 6 relating the property of a graph class  $\mathcal{F}$  to be closed under taking subgraphs to the property of  $\equiv_{\mathcal{F}}$  to be preserved under taking full complements.

Since our definition of the homomorphism distinguishing closure involves only simple graph in order to be aligned with [32], Theorem 6 deviates slightly from the other results in Table 1. This is because the relations  $\equiv_{\mathcal{F}}$  and  $\equiv_{\text{cl}(\mathcal{F})}$  a priori coincide only on simple graphs and not necessarily on all graphs, a crucial point raised by a reviewer.

► **Theorem 6.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under deleting edges,
- (ii)  $\equiv_{\mathcal{F}}$  is preserved under taking full complements, i.e. for all simple graphs  $G$  and  $H$  it holds that  $G \equiv_{\mathcal{F}} H$  if and only if  $\widehat{G} \equiv_{\mathcal{F}} \widehat{H}$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is closed under deleting edges,
- (iv)  $\text{cl}(\mathcal{F})$  is closed under taking subgraphs, i.e. it is closed under deleting edges and vertices,
- (v)  $\equiv_{\text{cl}(\mathcal{F})}$  is preserved under taking full complements,

the implications  $i \Rightarrow ii \Rightarrow iii \Leftrightarrow iv \Leftrightarrow v$  hold.

The linear expression required by Lemma 4 is provided by the following Lemma 7 based on the Inclusion–Exclusion principle. That iii implies iv follows from the Lemma 8. The proofs are deferred to the full version [35].

► **Lemma 7** ([24, Equation (5.23)]). *For every simple graph  $F$  and graph  $G$ ,*

$$\text{hom}(F, \widehat{G}) = \sum_{F' \subseteq F \text{ s.t. } V(F')=V(F)} (-1)^{|E(F')|} \text{hom}(F', G).$$

► **Lemma 8.** *If a homomorphism distinguishing closed graph class  $\mathcal{F}$  is closed under deleting edges then it is closed under taking subgraphs.*

### 3.3 Taking Minors and Preservation under Complements

The insights gained in the previous sections are now orchestrated to prove Theorem 9, which implies Theorem 2. This answers a question of Robertson [32, Question 8] affirmatively: Is it true that if  $\mathcal{F}$  is such that  $\equiv_{\mathcal{F}}$  is preserved under taking complements then there exist a minor-closed  $\mathcal{F}'$  such that  $\equiv_{\mathcal{F}}$  and  $\equiv_{\mathcal{F}'}$  coincide.

Theorem 9 is among the first results substantiating Robertson's conjecture not only for example classes but in full generality. In particular, as noted in [32, p. 2], there was little reason to believe that minor-closed graph families should play a distinct role in the theory of homomorphism indistinguishability. Theorem 9 indicates that this might be the case. Indeed, while Robertson's conjecture asserts that  $\text{cl}(\mathcal{F})$  coincides with  $\mathcal{F}$  for a minor-closed and union-closed graph class  $\mathcal{F}$ , Theorem 9 yields unconditionally that  $\text{cl}(\mathcal{F})$  is a minor-closed graph class itself.

► **Theorem 9.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under edge contraction and deletion,
- (ii)  $\equiv_{\mathcal{F}}$  is preserved under taking complements, i.e. for all simple graphs  $G$  and  $H$  it holds that  $G \equiv_{\mathcal{F}} H$  if and only if  $\widehat{G} \equiv_{\mathcal{F}} \widehat{H}$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is minor-closed,

*the implications  $i \Rightarrow ii \Leftrightarrow iii$  hold.*

Again, the strategy is to write  $\text{hom}(F, \widehat{G})$  as a linear combination of  $\text{hom}(F', G)$  for minors  $F'$  of  $F$ . For a simple graph  $G$ , write  $G^\circ$  for the graph obtained from  $G$  by adding a loop to every vertex. In light of Lemma 7, which concerns  $\text{hom}(F, \widehat{G})$ , noting that  $\widehat{G^\circ} \cong \widehat{G}$  for every simple graph  $G$ , it suffices to write  $\text{hom}(F, G^\circ)$  as a linear combination of  $\text{hom}(F', G)$  for minors  $F'$  of  $F$ . To that end, we consider a particular type of quotient graphs.

For a simple graph  $F$  and a set of edges  $L \subseteq E(F)$ , define the *contraction relation*  $\sim_L$  on  $V(F)$  by declaring  $v \sim_L w$  if  $v$  and  $w$  lie in the same connected component of the subgraph of  $F$  with vertex set  $V(F)$  and edge set  $L$ . Write  $[v]_L$  for the classes of  $v \in V(F)$  under the equivalence relation  $\sim_L$ .

The *contraction quotient*  $F \circledast L$  is the graph whose vertex set is the set of equivalence classes under  $\sim_L$  and with an edge between  $[v]_L$  and  $[w]_L$  if and only if there is an edge  $xy \in E(F) \setminus L$  such that  $x \sim_L v$  and  $y \sim_L w$ . In general,  $F \circledast L$  may contain loops, cf. Example 11. However, if it is simple then it is equal to  $F/\mathcal{P}$  where  $\mathcal{P}$  is the partition of  $V(F)$  into equivalence classes under  $\sim_L$ , i.e.  $\mathcal{P} := \{[v]_L \mid v \in V(F)\}$ . In this case,  $F \circledast L$  is a graph obtained from  $F$  by edge contractions.

With this notation, the quantity  $\text{hom}(F, G^\circ)$  can be succinctly written as linear combination. Using Proposition 10 and Lemma 7, Theorem 9 can be proven by the strategy described in Section 3.1. The proofs are deferred to [35].

► **Proposition 10.** *Let  $F$  and  $G$  be simple graphs. Then*

$$\text{hom}(F, G^\circ) = \sum_{L \subseteq E(F)} \text{hom}(F \circledast L, G).$$

► **Example 11.** Let  $K_3$  denote the clique with vertex set  $\{1, 2, 3\}$ . Then  $K_3 \circledast \emptyset \cong K_3$ ,  $K_3 \circledast \{12\} \cong K_2$ ,  $K_3 \circledast \{12, 23\} \cong K_1$ , and  $K_3 \circledast \{12, 23, 13\} \cong K_1$ . For every simple graph  $G$ ,  $\text{hom}(K_3, G^\circ) = \text{hom}(K_3, G) + 3 \text{hom}(K_2, G) + \text{hom}(K_1, G)$  since  $\text{hom}(K_1^\circ, G) = 0$ .

### 3.4 Taking Induced Subgraphs, Contracting Edges, and Lexicographic Products

In this section, it is shown that a homomorphism distinguishing closed graph class is closed under taking induced subgraphs (contracting edges) if and only if its homomorphism indistinguishability relation is preserved under lexicographic products with a fixed graph from the left (from the right).

Examples for equivalence relations preserved under lexicographic products related to chromatic graph parameters are listed in Corollary 18. Further examples are of model-theoretic nature and stem from winning strategies of the Duplicator player in bijective pebble games [35].

► **Theorem 12.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under taking induced subgraphs,
- (ii)  $\equiv_{\mathcal{F}}$  is such that for all simple graphs  $G, H, H'$ , if  $H \equiv_{\mathcal{F}} H'$  then  $G \cdot H \equiv_{\mathcal{F}} G \cdot H'$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is closed under taking induced subgraphs.

*the implications  $i \Rightarrow ii \Leftrightarrow iii$  hold.*

An example for a lexicographic product from the right is the  $n$ -blow-up  $G \cdot \overline{K_n}$  of a graph  $G$ . It can be seen [24] that every homomorphism indistinguishability relation is preserved under blow-ups. Preservation under arbitrary lexicographic products from the right, however, is a non-trivial property corresponding to the associated graph class being closed under edge contractions:

► **Theorem 13.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under edge contractions,
- (ii)  $\equiv_{\mathcal{F}}$  is such that for all simple graphs  $G, G', H$ , if  $G \equiv_{\mathcal{F}} G'$  then  $G \cdot H \equiv_{\mathcal{F}} G' \cdot H$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is closed under edge contractions.

*the implications  $i \Rightarrow ii \Leftrightarrow iii$  hold.*

For the proofs, homomorphism counts  $\text{hom}(F, G \cdot H)$  are written as linear combinations of homomorphism counts  $\text{hom}(F', H)$  where  $F'$  ranges over induced subgraphs of  $F$  in the case of Theorem 12 and over graphs obtained from  $F'$  by contracting edges in the case of Theorem 13. The following succinct formula for counts of homomorphisms into a lexicographic product is derived in the full version [35]. It may be of independent interest.

► **Theorem 14.** *Let  $F, G$ , and  $H$  be simple graphs. Then*

$$\text{hom}(F, G \cdot H) = \sum_{\mathcal{R}} \text{hom}(F/\mathcal{R}, G) \text{hom}\left(\sum_{R \in \mathcal{R}} F[R], H\right)$$

*where the outer sum ranges over all  $\mathcal{R} \in \Pi(V(F))$  such that  $F[R]$  is connected for all  $R \in \mathcal{R}$ .*

Theorem 14 yields Theorems 12 and 13, which in turn imply together the following Corollary 15. The proofs are deferred to the full version [35].

► **Corollary 15.** *For a graph class  $\mathcal{F}$  and the assertions*

- (i)  $\mathcal{F}$  is closed under taking induced subgraphs and edge contractions,
- (ii)  $\equiv_{\mathcal{F}}$  is such that for all simple graphs  $G, G', H$ , and  $H'$ , if  $G \equiv_{\mathcal{F}} G'$  and  $H \equiv_{\mathcal{F}} H'$  then  $G \cdot H \equiv_{\mathcal{F}} G' \cdot H'$ ,
- (iii)  $\text{cl}(\mathcal{F})$  is closed under taking induced subgraphs and edge contractions.

*the implications  $i \Rightarrow ii \Leftrightarrow iii$  hold.*

As a final observation, the following Lemma 16 relates the property of being closed under edge contractions to the other closure properties in Table 1.

► **Lemma 16.** *If a homomorphism distinguishing closed graph class  $\mathcal{F}$  is closed under contracting edges then it is closed under taking summands.*

### 3.5 Applications

As applications of Theorems 6, 9, 12, and 13, we conclude, in the spirit of [3], that certain equivalence relations on graphs cannot be homomorphism distinguishing relations.

► **Corollary 17.** *Let  $\mathcal{F}$  be a non-empty graph class such that one of the following holds:*

- $\equiv_{\mathcal{F}}$  is preserved under complements, cf. Theorem 9,
- $\equiv_{\mathcal{F}}$  is preserved under full complements, cf. Theorem 6,
- $\equiv_{\mathcal{F}}$  is preserved under left lexicographic products, cf. Theorem 12, or
- $\equiv_{\mathcal{F}}$  is preserved under right lexicographic products, cf. Theorem 13.

*Then  $G \equiv_{\mathcal{F}} H$  implies that  $|V(G)| = |V(H)|$  for all graphs  $G$  and  $H$ .*

**Proof.** By Theorems 6, 9, 12, and 13,  $\mathcal{F}$  can be chosen to be closed under taking minors, subgraphs, induced subgraphs, or contracting edges. In any case,  $K_1 \in \mathcal{F}$  as  $\mathcal{F}$  is non-empty and hence  $|V(G)| = \text{hom}(K_1, G) = \text{hom}(K_1, H) = |V(H)|$ . ◀

As concrete examples, consider the following relations.

► **Corollary 18.** *There is no graph class  $\mathcal{F}$  satisfying any of the following assertions for all graphs  $G$  and  $H$ :*

- (i)  $G \equiv_{\mathcal{F}} H$  iff  $a(G) = a(H)$  where  $a$  denotes the order of the automorphism group,
- (ii)  $G \equiv_{\mathcal{F}} H$  iff  $\alpha(G) = \alpha(H)$  where  $\alpha$  denotes the size of the largest independent set,
- (iii)  $G \equiv_{\mathcal{F}} H$  iff  $\omega(G) = \omega(H)$  where  $\omega$  denotes the size of the largest clique,
- (iv)  $G \equiv_{\mathcal{F}} H$  iff  $\chi(G) = \chi(H)$  where  $\chi$  denotes the chromatic number.

**Proof.** The relation in Item i is preserved under taking complements. By [17, Theorem 1, Corollary p. 90], the relations in Items ii and iv are preserved under left lexicographic products. For Item iii, the same follows from [17, Theorem 1] observing that  $\overline{G} \cdot \overline{H} = \overline{G \cdot H}$  and  $\omega(G) = \alpha(\overline{G})$ . In each case, it is easy to exhibit a pair of graphs  $G$  and  $H$  in the same equivalence class with different number of vertices. By Corollary 17, none of the equivalence relations is a homomorphism indistinguishability relation. ◀

## 4 Equivalences over Self-Complementary Logics

In this section, Theorem 1 is derived from Theorem 9. The theorem applies to self-complementary logics, of which examples are given subsequently. Finally, a result from graph minor theory is used to relate logics on graphs to quantum isomorphism.

► **Theorem 1.** *Let  $(L, \models)$  be a self-complementary logic on graphs for which there exists a graph class  $\mathcal{F}$  such that two graphs  $G$  and  $H$  are homomorphism indistinguishable over  $\mathcal{F}$  if and only if they are  $L$ -equivalent. Then there exists a minor-closed graph class  $\mathcal{F}'$  whose homomorphism indistinguishability relation coincides with  $L$ -equivalence.*

## 82:10 Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors

**Proof.** It is shown that  $\equiv_{\mathcal{F}}$  is preserved under taking complements in the sense of Theorem 9. Suppose  $G \equiv_{\mathcal{F}} H$ . By assumption, for all  $\varphi \in \mathbf{L}$  it holds that  $G \models \varphi$  iff  $H \models \varphi$  and hence, by self-complementarity,

$$\overline{G} \models \varphi \iff G \models \overline{\varphi} \iff H \models \overline{\varphi} \iff \overline{H} \models \overline{\overline{\varphi}} \iff \overline{H} \models \varphi.$$

Here, the penultimate equivalence holds since  $H \models \varphi$  if and only if  $H \models \overline{\overline{\varphi}}$  for all  $\varphi$  and  $H$  by the definition of self-complementarity observing that  $\overline{\overline{H}} \cong H$ . Thus,  $\overline{G} \equiv_{\mathcal{F}} \overline{H}$ . By Theorem 9,  $\mathcal{F}' := \text{cl}(\mathcal{F})$  is minor-closed.  $\blacktriangleleft$

In particular, by Corollary 17, all  $\mathbf{L}$ -equivalent graphs  $G$  and  $H$  must have the same number of vertices unless  $\mathbf{L}$  is trivial in the sense that all graphs  $G$  and  $H$  are  $\mathbf{L}$ -equivalent.

A first example of a self-complementarity logic is first-order logic  $\text{FO}$  over the signature of graphs  $\{E\}$ . In order to establish this property, a formula  $\overline{\varphi} \in \text{FO}$  has to be constructed for every  $\varphi \in \text{FO}$  such that  $G \models \varphi$  iff  $\overline{G} \models \overline{\varphi}$  for all graphs  $G$ . Only subformulae  $Exy$  require non-trivial treatment:

► **Definition 19.** For every  $\varphi \in \text{FO}$ , define  $\overline{\varphi} \in \text{FO}$  inductively as follows:

1. if  $\varphi = Exy$  then  $\overline{\varphi} := \neg Exy \wedge (x \neq y)$ ,
2. if  $\varphi = \perp$  or  $\varphi = \top$  then  $\overline{\varphi} := \varphi$ , if  $\varphi = (x = y)$  then  $\overline{\varphi} := \varphi$ , if  $\varphi = \neg\psi$  then  $\overline{\varphi} := \overline{\overline{\psi}}$ , if  $\varphi = \psi \wedge \chi$  then  $\overline{\varphi} := \overline{\psi} \wedge \overline{\chi}$ , if  $\varphi = \psi \vee \chi$  then  $\overline{\varphi} := \overline{\psi} \vee \overline{\chi}$ , if  $\varphi = \exists x. \psi$  then  $\overline{\varphi} := \exists x. \overline{\psi}$ , and if  $\varphi = \forall x. \psi$  then  $\overline{\varphi} := \forall x. \overline{\psi}$ .

► **Lemma 20.** Let  $\varphi \in \text{FO}$  be a formula with  $k \geq 0$  free variables. Then for all simple graphs  $G$  with  $\mathbf{v} \in V(G)^k$  it holds that

$$G, \mathbf{v} \models \varphi \iff \overline{G}, \mathbf{v} \models \overline{\varphi}.$$

In particular,  $\text{FO}$  is self-complementary.

**Proof.** The proof is by induction on the structure of  $\varphi$ . If  $\varphi$  is  $Exy$ , observe that  $v_1 v_2 \in E(G)$  if and only if  $v_1 v_2 \notin E(\overline{G})$  and  $v_1 \neq v_2$ . In all other cases, the claim is purely syntactical.  $\blacktriangleleft$

Lemma 20 gives a purely syntactical criterion for a fragment  $\mathbf{L} \subseteq \text{FO}$  to be self-complementary. Indeed, if  $\overline{\varphi} \in \mathbf{L}$  as defined in Definition 19 for all  $\varphi \in \mathbf{L}$  then  $\mathbf{L}$  is self-complementary. Note that the operation in Definition 19 increases neither the number of variables nor affects the quantifiers in the formula. Thus, Lemma 20 automatically extends to fragments of  $\text{FO}$  defined by restricting the number of variables, order or number of quantifiers. For extensions of  $\text{FO}$ , Definition 19 can be easily extended, cf. [35]. This yields a rich realm of self-complementary logics, of which the following Example 21 lists only a selection.

- **Example 21.** The following logics on graphs are self-complementary. For every  $k, d \geq 0$ ,
- the  $k$ -variable and quantifier-depth- $d$  fragments  $\text{FO}^k$  and  $\text{FO}_d$  of  $\text{FO}$ ,
  - first-order logic with counting quantifiers  $\text{C}$  and its  $k$ -variable and quantifier-depth- $d$  fragments  $\text{C}^k$  and  $\text{C}_d$ ,
  - inflationary fixed-point logic  $\text{IFP}$ , cf. [18],
  - second-order logic  $\text{SO}$  and its fragments monadic second-order logic  $\text{MSO}_1$ , existential second-order logic  $\text{ESO}$ , cf. [10, 16].

Corollary 17 readily gives an alternative proof of [3, Propositions 1 and 2], which assert that neither  $\text{FO}^k$ -equivalence nor  $\text{FO}_d$ -equivalence are characterised by homomorphism indistinguishability relations. The logic fragments  $\text{C}^k$  and  $\text{C}_d$  are however characterised by homomorphism indistinguishability relations [14, 19].

The final result of this section demonstrates how graph minor theory can yield insights into the expressive power of logics via Theorem 1. Subject to it are self-complementary logics which have a homomorphism indistinguishability characterisation and are stronger than  $C^k$  for every  $k$ , e.g. they are capable of distinguishing CFI-graphs [7]. It is shown that equivalence w.r.t. any such logic is a sufficient condition for quantum isomorphism, an undecidable equivalence comparing graphs [4].

► **Theorem 22.** *Let  $(L, \equiv)$  be a self-complementary logic on graphs for which there exists a graph class  $\mathcal{F}$  such that two graphs  $G$  and  $H$  are homomorphism indistinguishable over  $\mathcal{F}$  if and only if they are  $L$ -equivalent. Suppose that for all  $k \in \mathbb{N}$  there exist graphs  $G$  and  $H$  such that  $G \equiv_{C^k} H$  and  $G \not\equiv_L H$ . Then all  $L$ -equivalent graphs are quantum isomorphic.*

**Proof.** Contrapositively, it is shown that if there exist non-quantum-isomorphic  $L$ -equivalent graphs then there exists a  $k \in \mathbb{N}$  such that  $G \equiv_{C^k} H \implies G \equiv_L H$  for all  $G$  and  $H$ . By [25, 14], this statement can be rephrased in the language of homomorphism indistinguishability as  $\mathcal{P} \not\subseteq \text{cl}(\mathcal{F}) \implies \exists k \in \mathbb{N}. \mathcal{F} \subseteq \text{cl}(\mathcal{TW}_k)$  where  $\mathcal{P}$  denotes the class of all planar graphs and  $\mathcal{TW}_k$  the class of all graphs of treewidth at most  $k$ . By Theorem 9,  $\text{cl}(\mathcal{F})$  is a minor-closed graph class. By [34, (2.1)], cf. [29, Theorem 3.8], if  $\text{cl}(\mathcal{F})$  does not contain all planar graphs then it is of bounded treewidth. Hence, there exists a  $k \in \mathbb{N}$  such that  $\mathcal{F} \subseteq \text{cl}(\mathcal{F}) \subseteq \mathcal{TW}_k \subseteq \text{cl}(\mathcal{TW}_k)$ . ◀

## 5 Classification of Homomorphism Distinguishing Closed Essentially Profinite Graph Classes

The central result of this section is a classification of the homomorphism distinguishing closed graph classes which are in a sense finite. Since every homomorphism distinguishing closed graph class is closed under disjoint unions, infinite graph classes arise naturally when studying the semantic properties of the homomorphism indistinguishability relations of finite graph classes. Nevertheless, the infinite graph classes arising in this way are *essentially finite*, i.e. they exhibit only finitely many distinct connected components. One may generalise this definition slightly by observing that all graphs  $F$  admitting a homomorphism into some fixed graph  $G$  have chromatic number bounded by the chromatic number of  $G$ . Thus, in order to make a graph class  $\mathcal{F}$  behave much like an essentially finite class, it suffices to impose a finiteness condition, for every graph  $K$ , on the subfamily of all  $K$ -colourable graphs in  $\mathcal{F}$ .

Formally, for a graph  $F$ , write  $\Gamma(F)$  for the set of connected components of  $F$ . For a graph class  $\mathcal{F}$ , define  $\Gamma(\mathcal{F})$  as the union of the  $\Gamma(F)$  where  $F \in \mathcal{F}$ . For a graph class  $\mathcal{F}$  and a graph  $K$ , define  $\mathcal{F}_K := \{F \in \mathcal{F} \mid \text{hom}(F, K) > 0\}$ , the set of  $K$ -colourable graphs in  $\mathcal{F}$ .

► **Definition 23.** *A graph class  $\mathcal{F}$  is essentially finite if  $\Gamma(\mathcal{F})$  is finite. It is essentially profinite if  $\mathcal{F}_K$  is essentially finite for all graphs  $K$ .*

Clearly, every finite graph class is essentially finite and hence essentially profinite. Other examples for essentially profinite classes are the class of all cliques. They represent a special case of the following construction from [32, Theorem 6.16]: For every  $S \subseteq \mathbb{N}$ , the family

$$K^S := \{K_{n_1} + \dots + K_{n_r} \mid r \in \mathbb{N}, \{n_1, \dots, n_r\} \subseteq S\} \quad (5)$$

is essentially profinite. In particular, there are uncountably many such families of graphs. Note that one may replace the sequence of cliques  $(K_n)_{n \in \mathbb{N}}$  in Equation (5) by any other sequence of connected graphs  $(F_n)_{n \in \mathbb{N}}$  such that the sequence of chromatic numbers  $(\chi(F_n))_{n \in \mathbb{N}}$  takes every value only finitely often.

Every graph  $F$  of an essentially finite family  $\mathcal{F}$  can be represented uniquely as vector  $\vec{F} \in \mathbb{R}^{\Gamma(\mathcal{F})}$  whose  $C$ -th entry for  $C \in \Gamma(\mathcal{F})$  is the number of times the graph  $C$  appears as a connected component in  $F$ . The classification of the homomorphism distinguishing closed essentially profinite graph classes can now be stated as follows. The proof, deferred to the full version [35], is based on a generalisation of a result by Kwiecień, Marcinkowski, and Ostropolski-Nalewaja [22].

► **Theorem 24.** *For an essentially profinite graph class  $\mathcal{F}$ , the following are equivalent:*

- (i)  $\mathcal{F}$  is homomorphism distinguishing closed,
- (ii) For every graph  $K$ , if  $\vec{K} \in \text{span}\{\vec{F} \in \mathbb{R}^{\Gamma(\mathcal{F}_K \cup \{K\})} \mid F \in \mathcal{F}_K\}$  then  $K \in \mathcal{F}$ ,
- (iii)  $\mathcal{F}_K$  is homomorphism distinguishing closed for every graph  $K$ .

Theorem 24 directly implies Theorem 3. Indeed, if  $\mathcal{F}$  is union-closed and closed under summands then  $\Gamma(\mathcal{F}) \subseteq \mathcal{F}$  and every graph  $K$  such that  $\Gamma(K) \subseteq \Gamma(\mathcal{F})$  is itself in  $\mathcal{F}$ . In particular, Theorem 24 implies that all essentially profinite union-closed minor-closed graph classes are homomorphism distinguishing closed. For example, for every graph  $G$ , the union-closure of the class of minors of  $G$  is homomorphism distinguishing closed.

To demonstrate the inner workings of condition ii in Theorem 24, we consider the following examples. The first example shows that not even the weakest closure property from Figure 1 is shared by all homomorphism distinguishing closed families. The second example answers a question from [32, p. 29] negatively: Is the disjoint union closure of the union of homomorphism distinguishing closed families homomorphism distinguishing closed?

► **Example 25.** Let  $F_1, F_2$  be connected non-isomorphic graphs such that  $\text{hom}(F_1, F_2) > 0$ .

1. The class  $\mathcal{F}_1 := \{n(F_1 + F_2) \mid n \geq 1\}$  is homomorphism distinguishing closed and not closed under taking summands.
2. For the homomorphism distinguishing closed  $\mathcal{F}_2 := \{nF_1 \mid n \geq 1\}$ , the disjoint union closure of  $\mathcal{F}_1 \cup \mathcal{F}_2$  is not homomorphism distinguishing closed.

Some further enjoyable properties of essentially (pro)finite graph classes merit being commented on. Statements and proofs are deferred to the full version [35]. Firstly, the homomorphism indistinguishability relation of no essentially profinite graph class is as fine as the isomorphism relation  $\cong$ . This is because the homomorphism distinguishing closure of an essentially profinite graph class is essentially profinite. Secondly, the complexity-theoretic landscape of the problem  $\text{HOMIND}(\mathcal{F})$  of deciding whether two graphs  $G$  and  $H$  are homomorphism indistinguishable over an essentially profinite class  $\mathcal{F}$  is rather diverse. For essentially finite  $\mathcal{F}$ ,  $\text{HOMIND}(\mathcal{F})$  is in polynomial time. For essentially profinite  $\mathcal{F}$ , the problem can be arbitrarily hard.

## 6 Conclusion

The main technical contribution of this work is a characterisation of closure properties of graph classes  $\mathcal{F}$  in terms of preservation properties of their homomorphism indistinguishability relations  $\equiv_{\mathcal{F}}$ , cf. Table 1. In consequence, a surprising connection between logical equivalences and homomorphism indistinguishability over minor-closed graph classes is established. In this way, results from graph minor theory are made available to the study of the expressive power of logics on graphs. Finally, a full classification of the homomorphism distinguishing closed graph classes which are essentially profinite is given. Various open questions of [32] are answered by results clarifying the properties of homomorphism indistinguishability relations and of the homomorphism distinguishing closure.

It is tempting to view the results in Table 1 as instances of a potentially richer connection between graph-theoretic properties of  $\mathcal{F}$  and *polymorphisms* of  $\equiv_{\mathcal{F}}$ , i.e. isomorphism-invariant maps  $f$  sending tuples of graphs to graphs such that  $f(G_1, \dots, G_k) \equiv_{\mathcal{F}} f(H_1, \dots, H_k)$  whenever  $G_i \equiv_{\mathcal{F}} H_i$  for all  $i \in [k]$ . Recalling the algebraic approach to CSPs, cf. [5], one may ask what structural insights into  $\mathcal{F}$  can be gained by considering polymorphisms of  $\equiv_{\mathcal{F}}$ . More concretely, can bounded treewidth or closure under topological minors be characterised in terms of some polymorphism?

---

## References

- 1 Samson Abramsky, Tomáš Jakl, and Thomas Paine. Discrete density comonads and graph parameters. In Helle Hvid Hansen and Fabio Zanasi, editors, *Coalgebraic Methods in Computer Science*, pages 23–44. Springer International Publishing, 2022. doi:10.1007/978-3-031-10736-8\_2.
- 2 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, 2008*, pages 241–249, 2008. doi:10.1093/bioinformatics/btn163.
- 3 Albert Atserias, Phokion G. Kolaitis, and Wei-Lin Wu. On the Expressive Power of Homomorphism Counts. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470543.
- 4 Albert Atserias, Laura Mančinska, David E. Roberson, Robert Šámal, Simone Severini, and Antonios Varvitsiotis. Quantum and non-signalling graph isomorphisms. *J. Comb. Theory, Ser. B*, 136:289–328, 2019. doi:10.1016/j.jctb.2018.11.002.
- 5 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and How to Use Them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol17.15301.1.
- 6 Paul Beaujean, Florian Sikora, and Florian Yger. Graph Homomorphism Features: Why Not Sample? In Michael Kamp, Irena Koprinska, Adrien Bibal, Tassadit Bouadi, Benoît Frénay, Luis Galárraga, José Oramas, Linara Adilova, Yamuna Krishnamurthy, Bo Kang, Christine Largeron, Jeffrey Lijffijt, Tiphaine Viard, Pascal Welke, Massimiliano Ruocco, Erlend Aune, Claudio Gallicchio, Gregor Schiele, Franz Pernkopf, Michaela Blott, Holger Fröning, Günther Schindler, Riccardo Guidotti, Anna Monreale, Salvatore Rinzivillo, Przemyslaw Biecek, Eirini Ntoutsis, Mykola Pechenizkiy, Bodo Rosenhahn, Christopher Buckley, Daniela Cialfi, Pablo Lanillos, Maxwell Ramstead, Tim Verbelen, Pedro M. Ferreira, Giuseppina Andresini, Donato Malerba, Ibéria Medeiros, Philippe Fournier-Viger, M. Saqib Nawaz, Sebastian Ventura, Meng Sun, Min Zhou, Valerio Bitetta, Ilaria Bordino, Andrea Ferretti, Francesco Gullo, Giovanni Ponti, Lorenzo Severini, Rita Ribeiro, João Gama, Ricard Gavaldà, Lee Cooper, Naghme Ghazaleh, Jonas Richiardi, Damian Roqueiro, Diego Saldana Miranda, Konstantinos Sechidis, and Guilherme Graça, editors, *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, volume 1524, pages 216–222. Springer International Publishing, Cham, 2021. Series Title: Communications in Computer and Information Science. doi:10.1007/978-3-030-93736-2\_17.
- 7 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, December 1992. doi:10.1007/BF01305232.
- 8 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of Real Conjunctive Queries. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 59–70. ACM Press, 1993. doi:10.1145/153850.153856.

- 9 Yijia Chen, Jörg Flum, Mingjun Liu, and Zhiyang Xun. On algorithms based on finitely many homomorphism counts. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPICs*, pages 32:1–32:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.MFCS.2022.32.
- 10 Bruno Courcelle. The monadic second order logic of graphs VI: on several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2):117–149, 1994. doi:10.1016/0166-218X(94)90019-1.
- 11 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms Are a Good Basis for Counting Small Subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 210–223. Association for Computing Machinery, 2017. event-place: Montreal, Canada. doi:10.1145/3055399.3055502.
- 12 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-Type Theorems and Game Comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.
- 13 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 40:1–40:14, 2018. doi:10.4230/LIPICs.ICALP.2018.40.
- 14 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, August 2010. doi:10.1002/jgt.20461.
- 15 H.-D. Ebbinghaus. Extended logics: The general framework. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Logic, pages 25–76. Cambridge University Press, 2017. doi:10.1017/9781316717158.005.
- 16 Thomas Eiter, Georg Gottlob, and Thomas Schwentick. The Model Checking Problem for Prefix Classes of Second-Order Logic: A Survey. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation*, volume 6300, pages 227–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-15025-8\_13.
- 17 Dennis Geller and Saul Stahl. The chromatic number and other functions of the lexicographic product. *Journal of Combinatorial Theory, Series B*, 19(1):87–95, 1975. doi:10.1016/0095-8956(75)90076-3.
- 18 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017. doi:10.1017/9781139028868.
- 19 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '20*, pages 507–520, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394739.
- 20 Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.
- 21 Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020. doi:10.1007/s41109-019-0195-3.
- 22 Jarosław Kwiecień, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Determinacy of Real Conjunctive Queries. The Boolean Case. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 347–358. Association for Computing Machinery, 2022. doi:10.1145/3517804.3524168.
- 23 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3):321–328, September 1967. doi:10.1007/BF02280291.
- 24 László Lovász. *Large networks and graph limits*. Number volume 60 in American Mathematical Society colloquium publications. American Mathematical Society, Providence, Rhode Island, 2012. doi:10.1090/coll/060.

- 25 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. doi:10.1109/FOCS46700.2020.00067.
- 26 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.
- 27 Yoàv Montacute and Nihil Shah. The Pebble-Relation Comonad in Finite Model Theory. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 13:1–13:11. ACM, 2022. doi:10.1145/3531130.3533335.
- 28 Daniel Neuen. Homomorphism-Distinguishing Closedness for Graphs of Bounded Tree-Width, April 2023. doi:10.48550/arXiv.2304.07011.
- 29 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Number 28 in Algorithms and combinatorics. Springer, Heidelberg ; New York, 2012. doi:10.1007/978-3-642-27875-4.
- 30 Hoang Nguyen and Takanori Maehara. Graph homomorphism convolution. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7306–7316. PMLR, 2020. URL: <http://proceedings.mlr.press/v119/nguyen20c.html>.
- 31 Gaurav Rattan and Tim Seppelt. Weisfeiler–Leman and Graph Spectra. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2268–2285, 2023. doi:10.1137/1.9781611977554.ch87.
- 32 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree, June 2022. doi:10.48550/arXiv.2206.10321.
- 33 David E. Roberson and Tim Seppelt. Lasserre Hierarchy for Graph Isomorphism and Homomorphism Indistinguishability. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 101:1–101:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.101.
- 34 Neil Robertson and P.D Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, August 1986. doi:10.1016/0095-8956(86)90030-4.
- 35 Tim Seppelt. Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors, 2023. doi:10.48550/arXiv.2302.11290.



# Decomposing Finite Languages

Daniel Alexander Spenner  

Technische Universität Dortmund, Germany

---

## Abstract

The paper completely characterizes the primality of acyclic DFAs, where a DFA  $\mathcal{A}$  is *prime* if there do not exist DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  with  $\mathcal{L}(\mathcal{A}) = \bigcap_{i=1}^t \mathcal{L}(\mathcal{A}_i)$  such that each  $\mathcal{A}_i$  has strictly less states than the minimal DFA recognizing the same language as  $\mathcal{A}$ . A regular language is prime if its minimal DFA is prime. Thus, this result also characterizes the primality of finite languages.

Further, the NL-completeness of the corresponding decision problem  $\text{PRIME-DFA}_{\text{fin}}$  is proven. The paper also characterizes the primality of acyclic DFAs under two different notions of compositionality, union and union-intersection compositionality.

Additionally, the paper introduces the notion of *S-primality*, where a DFA  $\mathcal{A}$  is S-prime if there do not exist DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  with  $\mathcal{L}(\mathcal{A}) = \bigcap_{i=1}^t \mathcal{L}(\mathcal{A}_i)$  such that each  $\mathcal{A}_i$  has strictly less states than  $\mathcal{A}$  itself. It is proven that the problem of deciding S-primality for a given DFA is NL-hard. To do so, the NL-completeness of  $2\text{MINIMAL-DFA}$ , the basic problem of deciding minimality for a DFA with at most two letters, is proven.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Regular languages; Theory of computation  $\rightarrow$  Problems, reductions and completeness

**Keywords and phrases** Deterministic finite automaton (DFA), Regular languages, Finite languages, Decomposition, Primality, Minimality

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.83

**Related Version** The full version includes the appendix, which contains proofs for all results.

*Full Version:* <https://arxiv.org/abs/2307.06802>

**Acknowledgements** I want to thank Thomas Schwentick for his advice and encouragement.

## 1 Introduction

Under intersection compositionality a deterministic finite automaton (DFA)  $\mathcal{A}$  is *composite* if there exist DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  with  $\mathcal{L}(\mathcal{A}) = \bigcap_{i=1}^t \mathcal{L}(\mathcal{A}_i)$  such that the size of each  $\mathcal{A}_i$  is smaller than the index of  $\mathcal{A}$ . Otherwise,  $\mathcal{A}$  is *prime* [10]. The index of  $\mathcal{A}$  is the size of the minimal DFA recognizing the same language as  $\mathcal{A}$ .  $\text{PRIME-DFA}$  denotes the problem of deciding primality for a given DFA.  $\text{PRIME-DFA}_{\text{fin}}$  denotes the restriction of  $\text{PRIME-DFA}$  to DFAs recognizing a finite language.

Compositionality in general is a key concept in both practical and theoretical computer science [3, 16]. The intersection decomposition of finite automata can be motivated by LTL model checking as well as automaton identification. Both will be briefly discussed below.

The notion of intersection compositionality of finite automata was introduced in [10], while a limitation of this notion was already studied in [5]. Surprisingly, [10] found even the complexity of the basic problem  $\text{PRIME-DFA}$  to be open. They proved that  $\text{PRIME-DFA}$  is in  $\text{EXPSPACE}$  and is NL-hard. So far, this doubly exponential gap has not been closed.

Given the difficulties in tackling the general problem, it has proven fruitful to characterize the intersection compositionality of fragments of the regular languages [10, 8, 9]. Our study joins this line of research by completely characterizing the intersection compositionality of acyclic DFAs (ADFA) and thereby of finite languages. Further, we prove the NL-completeness of  $\text{PRIME-DFA}_{\text{fin}}$  and characterize the compositionality of finite languages under two different notions of compositionality suggested in [10], union and union-intersection compositionality.



© Daniel Alexander Spenner;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 83; pp. 83:1–83:14

Leibniz International Proceedings in Informatics



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

Additionally, we present a proof of the NL-completeness of the basic problem 2MINIMAL-DFA, the problem of deciding minimality for a DFA with at most two letters. For arbitrary alphabets, the NL-hardness is a folklore result that seemingly has not been explicitly published but follows from the constructions in [2], while the NL-hardness of 2MINIMAL-DFA appears to be new [4]. We use this result to establish complexity boundaries for S-PRIME-DFA, a modification of PRIME-DFA using the size of the given DFA, not its index.

**Related Work.** The notion of intersection compositionality was introduced in [10], where the aforementioned complexity boundaries were established. They already considered language fragments, analyzing safety DFAs and permutation DFAs. This line of research was followed up in [8, 9], which focused on unary DFAs and permutation DFAs, respectively.

The intersection decomposition of automata can be motivated by LTL model checking, where the validity of a specification, given as an LTL formula, is checked for a system. The automata-based approach entails translating the specification into a finite automaton [17]. Since the LTL model checking problem is PSPACE-complete in the size of the LTL formula [1], it is desirable to decompose the formula into a conjunction of subformulas. This can also be understood as decomposing the finite automaton corresponding to the formula.

Another application of intersection decomposition arises in the field of automaton identification. The basic task here is, given a set of labeled words, to construct a finite automaton conforming to this set [6]. An interesting approach is to construct multiple automata instead of one, which can lead to smaller and more intuitive solutions [11].

An alternative notion of compositionality uses concatenation. Here, a language  $L$  is composite if there exist two non-trivial languages  $L_1, L_2$  with  $L = L_1L_2$ . The concatenation primality problem for regular languages is PSPACE-complete [12]. The restriction to finite languages is known to be NP-hard [15], while the conjectured NP-completeness of this restriction remains open [14, 13, 18].

**Contributions.** In Section 3 we completely characterize the intersection compositionality of ADFAs and thereby of finite languages. We expand on this by proving the NL-completeness of PRIME-DFA<sub>fin</sub> in Section 4, thus showing that finite languages are significantly easier to handle under intersection compositionality than under concatenation compositionality. We characterize the union and union-intersection compositionality of finite languages in Section 5, where we also prove the existence of languages that are union-intersection composite but both union prime and intersection prime.

In Section 6 we introduce the problem S-PRIME-DFA, which is analogous to PRIME-DFA but uses the size for the definition of compositionality, not the index. We prove that S-PRIME-DFA is in EXPSpace and is NL-hard. We also prove these boundaries for 2PRIME-DFA and 2S-PRIME-DFA, the restrictions of the respective problems to DFAs with at most two letters. To establish these boundaries we prove the NL-completeness of 2MINIMAL-DFA.

Detailed proofs of these results are provided in the appendix.

## 2 Preliminaries

A *deterministic finite automaton* (DFA) is a 5-tuple  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ , where  $Q$  is a finite set of states,  $\Sigma$  is a finite non-empty alphabet,  $q_I \in Q$  is an initial state,  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function, and  $F \subseteq Q$  is a set of accepting states. As usual, we extend  $\delta$  to words:  $\delta : Q \times \Sigma^* \rightarrow Q$  with  $\delta(q, \varepsilon) = q$  and  $\delta(q, \sigma_1 \dots \sigma_n) = \delta(\delta(q, \sigma_1 \dots \sigma_{n-1}), \sigma_n)$ . For  $q \in Q$ , the DFA  $\mathcal{A}^q$  is constructed out of  $\mathcal{A}$  by setting  $q$  as the initial state, thus  $\mathcal{A}^q = (Q, \Sigma, q, \delta, F)$ .

The *run* of  $\mathcal{A}$  on a word  $w = \sigma_1 \dots \sigma_n$  starting in state  $q$  is the sequence  $q_0, \sigma_1, q_1, \dots, \sigma_n, q_n$  with  $q_0 = q$  and  $q_i = \delta(q_{i-1}, \sigma_i)$  for each  $i \in \{1, \dots, n\}$ . The *initial run* of  $\mathcal{A}$  on  $w$  is the run of  $\mathcal{A}$  on  $w$  starting in  $q_I$ . The run of  $\mathcal{A}$  on  $w$  starting in  $q$

is *accepting* if  $q_n \in F$ , otherwise it is *rejecting*. The DFA  $\mathcal{A}$  *accepts*  $w$  if the initial run of  $\mathcal{A}$  on  $w$  is accepting. Otherwise, it *rejects*  $w$ . The *language*  $\mathcal{L}(\mathcal{A})$  of  $\mathcal{A}$  is the set of words accepted by  $\mathcal{A}$ . We say that  $\mathcal{A}$  *recognizes*  $\mathcal{L}(\mathcal{A})$ . A language is *regular* if there exists a DFA recognizing it. Since we only consider regular languages, we use the terms language and regular language interchangeably.

The *size*  $|\mathcal{A}|$  of  $\mathcal{A}$  is the number of states in  $Q$ . The DFA  $\mathcal{A}$  is *minimal* if  $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{B})$  holds for every DFA  $\mathcal{B}$  with  $|\mathcal{B}| < |\mathcal{A}|$ . It is well known that for every regular language  $L$  there exists a canonical minimal DFA recognizing  $L$ . The *index*  $\text{ind}(L)$  of  $L$  is the size of this canonical minimal DFA. The index of  $\mathcal{A}$  is the index of the language recognized by  $\mathcal{A}$ , thus  $\text{ind}(\mathcal{A}) = \text{ind}(\mathcal{L}(\mathcal{A}))$ . Note that  $\mathcal{A}$  is minimal iff  $|\mathcal{A}| = \text{ind}(\mathcal{A})$ .

We borrow a few terms from graph theory. Let  $q_0, \sigma_1, q_1, \dots, \sigma_n, q_n$  be the run of  $\mathcal{A}$  on  $w = \sigma_1 \dots \sigma_n$  starting in  $q_0$ . Then  $q_0, \dots, q_n$  is a *path* in  $\mathcal{A}$  from  $q_0$  to  $q_n$ . The *length* of this path is  $n$ . Thus, for two states  $q, q'$  there exists a path from  $q$  to  $q'$  in  $\mathcal{A}$  of length  $n$  iff there exists a  $w \in \Sigma^n$  with  $\delta(q, w) = q'$ . The state  $q'$  is *reachable from*  $q$  if there exists a path from  $q$  to  $q'$ . Otherwise,  $q'$  is *unreachable from*  $q$ . Obviously, if  $q'$  is reachable from  $q$  then there exists a path from  $q$  to  $q'$  of a length strictly smaller than  $|\mathcal{A}|$ . We say that  $q'$  is *reachable* if it is reachable from  $q_I$ . Otherwise, it is *unreachable*. A *cycle* in  $\mathcal{A}$  is a path  $q_0, \dots, q_n$  in  $\mathcal{A}$  where  $q_0 = q_n$  and  $n \in \mathbb{N}_{\geq 1}$ . The DFA  $\mathcal{A}$  is *acyclic* (ADFA) if every cycle in  $\mathcal{A}$  begins in a rejecting sink. Clearly, a DFA recognizes a finite language iff its minimal DFA is acyclic.

We call a DFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  *linear* if for every  $q, q' \in Q$  with  $q \neq q'$  either  $q'$  is reachable from  $q$  or  $q$  is reachable from  $q'$ , but not both. Thus, in a linear DFA reachability induces a linear order over the states. Obviously, every linear DFA has exactly one sink. Furthermore, a minimal ADFA  $\mathcal{A}$  is linear iff  $|\mathcal{A}| = n + 2$ , where  $n$  is the length of the longest word in  $\mathcal{L}(\mathcal{A})$ .

Consider a word  $w = \sigma_1 \dots \sigma_n \in \Sigma^n$ . A word  $wv$  with  $v \in \Sigma^+$  is an *extension* of  $w$ . A word  $\sigma_1 \dots \sigma_i \sigma_{i+l} \dots \sigma_n$  with  $i \in \{0, \dots, n-2\}, l \in \{2, \dots, n-i\}$  is a *compression* of  $w$ . An ADFA  $\mathcal{A}$  has the *compression-extension-property* (CEP) if for every  $w \in \mathcal{L}(\mathcal{A})$  with  $|w| = n$ , where  $n$  is the length of the longest word in  $\mathcal{L}(\mathcal{A})$ , there exists a compression  $w'$  of  $w$  such that every extension of  $w'$  is rejected by  $\mathcal{A}$ .

We introduce a type of DFA already inspected in [10]. A regular language  $L \subseteq \Sigma^*$  is a *safety language* if  $w \notin L$  implies  $wy \notin L$  for every  $y \in \Sigma^*$ . A DFA  $\mathcal{A}$  is a *safety DFA* if  $\mathcal{L}(\mathcal{A})$  is a safety language. A regular language  $L \subseteq \Sigma^*$  is a *co-safety language* if the complement language  $\bar{L}$  of  $L$  is a safety language. A DFA  $\mathcal{A}$  is a *co-safety DFA* if  $\mathcal{L}(\mathcal{A})$  is a co-safety language. Clearly, every non-trivial minimal safety DFA has exactly one rejecting state, and this state is a sink. Conversely, every non-trivial minimal co-safety DFA has exactly one accepting state, and this state is a sink.

We introduce the notions of intersection compositionality and primality of DFAs and languages, following the definitions in [10]:

► **Definition 2.1.** For  $k \in \mathbb{N}_{\geq 1}$ , a DFA  $\mathcal{A}$  is *k-decomposable* if there exist DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  with  $\mathcal{L}(\mathcal{A}) = \bigcap_{i=1}^t \mathcal{L}(\mathcal{A}_i)$  and  $|\mathcal{A}_i| \leq k$  for each  $i \in \{1, \dots, t\}$ , where  $t \in \mathbb{N}_{\geq 1}$ . We call such DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  a *k-decomposition* of  $\mathcal{A}$ . We call  $\mathcal{A}$  *composite* if  $\mathcal{A}$  is *k-decomposable* for a  $k < \text{ind}(\mathcal{A})$ , that is, if it is  $(\text{ind}(\mathcal{A}) - 1)$ -decomposable. Otherwise, we call  $\mathcal{A}$  *prime*. ◻

We use compositionality or  $\cap$ -compositionality when referring to intersection compositionality.

When analyzing the compositionality of a given DFA  $\mathcal{A}$ , it is sufficient to consider minimal DFAs  $\mathcal{B}$  strictly smaller than the minimal DFA of  $\mathcal{A}$  with  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ . Thus, we define  $\alpha(\mathcal{A}) = \{\mathcal{B} \mid \mathcal{B} \text{ is a minimal DFA with } \text{ind}(\mathcal{B}) < \text{ind}(\mathcal{A}) \text{ and } \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})\}$ . Obviously, the DFA  $\mathcal{A}$  is composite iff  $\mathcal{L}(\mathcal{A}) = \bigcap_{\mathcal{B} \in \alpha(\mathcal{A})} \mathcal{L}(\mathcal{B})$ . We call a word  $w \in (\bigcap_{\mathcal{B} \in \alpha(\mathcal{A})} \mathcal{L}(\mathcal{B})) \setminus \mathcal{L}(\mathcal{A})$  a *primality witness* of  $\mathcal{A}$ . Clearly, the DFA  $\mathcal{A}$  is composite iff  $\mathcal{A}$  has no primality witness.

We extend the notions of  $k$ -decompositions, compositionality, primality and primality witnesses to regular languages by identifying a regular language with its minimal DFA.

We denote the problem of deciding primality for a given DFA with PRIME-DFA. We denote the restriction of PRIME-DFA to DFAs recognizing finite languages with PRIME-DFA<sub>fin</sub>. PRIME-DFA is in EXPSPACE and is NL-hard [10].

We denote the connectivity problem in directed graphs, which is NL-complete [7], with STCON. We denote the restriction of STCON to graphs with a maximum outdegree of two with 2STCON. Clearly, 2STCON is NL-complete as well. We denote the problem of deciding minimality for a given DFA with MINIMAL-DFA. For  $k \in \mathbb{N}_{\geq 2}$ , the problem  $k$ MINIMAL-DFA is the restriction of MINIMAL-DFA to DFAs with at most  $k$  letters. As mentioned in Section 1, the NL-completeness of  $k$ MINIMAL-DFA for  $k \in \mathbb{N}_{\geq 3}$  is folklore, while the NL-hardness of 2MINIMAL-DFA appears to be open.

### 3 Compositionality of Finite Languages

We characterize the compositionality of ADFAs and thereby of finite languages by proving:

► **Theorem 3.1.** *Consider a minimal ADFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  recognizing a non-empty language. Then  $\mathcal{A}$  is prime iff  $\mathcal{A}$  is linear and:*

- (i)  $\sigma^n \in \mathcal{L}(\mathcal{A})$  for some  $\sigma \in \Sigma$ , where  $n \in \mathbb{N}$  is the length of the longest word in  $\mathcal{L}(\mathcal{A})$ , or
- (ii)  $\mathcal{A}$  is a safety DFA and  $\mathcal{A}$  does not have the CEP. ┘

To prove Theorem 3.1 we will consider five cases in turn.

First, if the ADFA  $\mathcal{A}$  is not linear we essentially have a surplus of states, allowing us to construct one DFA rejecting overlong words and one specific DFA for each of the remaining words also rejected by  $\mathcal{A}$ . This approach fails with linear ADFAs. Nevertheless, we will come back to the idea of excluding words longer than a threshold value and tailoring a DFA for each word shorter than the threshold value which has to be rejected as well.

Second, if  $\mathcal{A}$  is linear and  $\sigma^n \in \mathcal{L}(\mathcal{A})$  holds the DFAs in  $\alpha(\mathcal{A})$  do not possess enough states to differentiate the words  $\sigma^0, \dots, \sigma^n$  but have to accept  $\sigma^n$ , which implies cyclic behavior on the words in  $\{\sigma\}^*$  from which primality follows.

Third, if there is no  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  and  $\mathcal{A}$  is not a safety DFA we can return to the idea of excluding words longer than a threshold value. For each of the words left to reject, it is possible to construct a DFA similar to  $\mathcal{A}$  but without the rejecting sink, which circles back to the rejecting non-sink.

Fourth, if there is no  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  and  $\mathcal{A}$  has the CEP we can utilize DFAs similar to  $\mathcal{A}$  possessing a rejecting sink, since the CEP allows us to skip over one state.

Fifth and finally, if  $\mathcal{A}$  is linear and  $\mathcal{A}$  is a safety DFA and does not have the CEP both of the above approaches fail. There is no state to circle back to, and for the word breaching the CEP skipping over states is not possible either, which implies primality.

Formalizing these five cases, we get:

▷ **Claim 3.2.** Consider a minimal ADFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  recognizing a non-empty language. Let  $n \in \mathbb{N}$  be the length of the longest word in  $\mathcal{L}(\mathcal{A})$ . The following assertions hold:

- (a)  $\mathcal{A}$  is composite if  $\mathcal{A}$  is not linear.
- (b)  $\mathcal{A}$  is prime if  $\mathcal{A}$  is linear and  $\sigma^n \in \mathcal{L}(\mathcal{A})$  holds for some  $\sigma \in \Sigma$ .
- (c)  $\mathcal{A}$  is composite if there is no  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  and  $\mathcal{A}$  is not a safety DFA.
- (d)  $\mathcal{A}$  is composite if there is no  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  and  $\mathcal{A}$  has the CEP.
- (e)  $\mathcal{A}$  is prime if  $\mathcal{A}$  is linear and  $\mathcal{A}$  is a safety DFA and  $\mathcal{A}$  does not have the CEP. ┘

Formalizing the intuition given above for (a) and (b) is not too complex. Assertions (c)–(e) prove to be much harder. Thus, we commence by discussing (c) in Section 3.1 and (d) and (e) in Section 3.2. Henceforth, we consider a minimal ADFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  recognizing the non-empty language  $L$  with  $\sigma^n \notin L$  for each  $\sigma \in \Sigma$ , where  $n \in \mathbb{N}$  is the length of the longest word in  $L$ . W.l.o.g. we assume  $Q = \{q_0, \dots, q_{n+1}\}$  with  $q_j$  being reachable from  $q_i$  for all  $i < j$ , which implies  $q_I = q_0$  and  $q_n \in F$  with  $q_{n+1}$  being the rejecting sink. Finally, we define  $\Sigma_{i,j} = \{\sigma \in \Sigma \mid \delta(q_i, \sigma) = q_j\}$ .

### 3.1 Linear non-safety ADFAs

We consider Claim 3.2 (c). Therefore, we assume that  $\mathcal{A}$  is not a safety DFA, which implies  $\{q_n\} \subseteq F \subset Q \setminus \{q_{n+1}\}$ . Let  $d \in \{0, \dots, n-1\}$  with  $q_d \notin F$ .

We show the compositionality of  $\mathcal{A}$  by specifying an  $(n+1)$ -decomposition of  $\mathcal{A}$ . First, we construct DFAs rejecting words not in  $L$  that are not extensions of words  $u \in L, |u| = n$ . Afterwards, we turn to such extensions, whose handling poses the main difficulty. Here, we first construct DFAs rejecting such extensions that are longer than a certain threshold value. For the remaining extensions we employ the idea of circling back to  $q_d$ .

We begin by considering words not in  $L$  which are not extensions of words  $u \in L, |u| = n$ . We introduce three DFA types handling these words.

First, let  $\mathcal{A}_0$  be the DFA constructed out of  $\mathcal{A}$  by removing  $q_n$ , redirecting every transition  $q \rightarrow q_n$  to  $q_0$ , and including  $q_0$  into the acceptance set. Clearly,  $\mathcal{A}_0 \in \alpha(\mathcal{A})$  and  $\mathcal{A}_0$  rejects every  $w \notin L$  on which  $\mathcal{A}$  enters the rejecting sink prematurely, that is, without entering  $q_n$ .

Second, let  $\hat{\mathcal{A}}_d$  be the DFA constructed out of  $\mathcal{A}$  by removing  $q_{n+1}$ , redirecting every transition  $q_i \rightarrow q_{n+1}$  with  $i < n$  to  $q_n$  and every transition  $q_n \rightarrow q_{n+1}$  to  $q_d$ . Clearly,  $\hat{\mathcal{A}}_d \in \alpha(\mathcal{A})$  and  $\hat{\mathcal{A}}_d$  rejects every  $w \notin L$  on which  $\mathcal{A}$  does not enter the rejecting sink.

Third, we construct DFAs rejecting extensions of words  $w \in L, |w| < n$  with  $\delta(q_0, w) = q_n$ . Let  $I = \{0, \dots, n\}$ . For each  $m \in \{1, \dots, n-1\}$  let  $I_m = \{(i_0, \dots, i_m) \in I^{m+1} \mid 0 = i_0 < \dots < i_m = n\}$ . For each  $\underline{i} \in I_m$  define  $\mathcal{A}_{\underline{i}}$  as in Figures 1a and 1b. It is easy to confirm that each  $\mathcal{A}_{\underline{i}}$  is in  $\alpha(\mathcal{A})$  and rejects extensions of words on which  $\mathcal{A}$  visits the states  $q_{i_0}, \dots, q_{i_m}$ .

Lemma 3.3 formalizes the results concerning  $\mathcal{A}_0, \hat{\mathcal{A}}_d$  and  $\mathcal{A}_{\underline{i}}$ :

► **Lemma 3.3.** *The following assertions hold:*

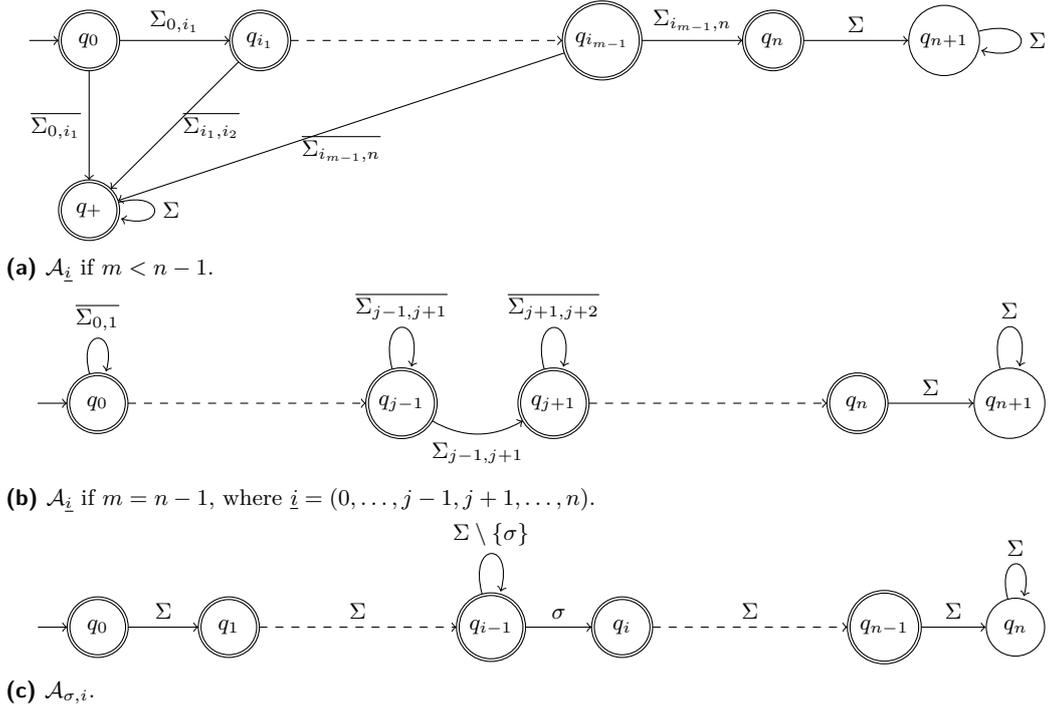
- (i)  $\mathcal{A}_0, \hat{\mathcal{A}}_d, \mathcal{A}_{\underline{i}} \in \alpha(\mathcal{A})$ , where  $\underline{i} \in \bigcup_{m=1}^{n-1} I_m$ .
- (ii) Consider a word  $w \notin L$ , where  $w$  is not an extension of a word  $u \in L, |u| = n$ . Then  $w \notin \mathcal{L}(\mathcal{A}_0) \cap \mathcal{L}(\hat{\mathcal{A}}_d) \cap \bigcap_{m=1}^{n-1} \bigcap_{\underline{i} \in I_m} \mathcal{L}(\mathcal{A}_{\underline{i}})$  holds. ◻

Next, we turn to the extensions of words  $u \in L, |u| = n$ . We begin by constructing DFAs that taken together reject every word strictly longer than  $n + (n-2)$ . Then we turn to the remaining extensions one by one, of which only a finite number are left to reject.

Let  $\sigma \in \Sigma$ . Since  $\sigma^n \notin L$ , there exists a value  $i \in \{1, \dots, n\}$  with  $\sigma \notin \Sigma_{i-1,i}$ . Define  $\mathcal{A}_{\sigma,i}$  as in Figure 1c. First, note that  $\mathcal{A}_{\sigma,i} \in \alpha(\mathcal{A})$  because a word rejected by  $\mathcal{A}_{\sigma,i}$  is strictly longer than  $n$  or is of length  $n$  with letter  $\sigma$  at position  $i$ . Next, consider a word  $w = \sigma_1 \dots \sigma_m \in \Sigma^m$  such that  $\sigma_j = \sigma$  for a  $j \in \{1, \dots, m\}$  with  $j \geq i$  and  $m \geq j + (n-i)$ . After reading the prefix  $\sigma_1 \dots \sigma_{j-1}$  the DFA  $\mathcal{A}_{\sigma,i}$  is at least in state  $q_{i-1}$ . Thus, after reading  $\sigma_1 \dots \sigma_j$  it is at least in state  $q_i$  and will reject after reading  $n-i$  more letters. Since  $m \geq j + (n-i)$ , we have  $w \notin \mathcal{L}(\mathcal{A}_{\sigma,i})$ . Lemma 3.4 formalizes this result:

► **Lemma 3.4.** *Let  $\sigma \in \Sigma$  and  $i \in \{1, \dots, n\}$  with  $\sigma \notin \Sigma_{i-1,i}$ . The following assertions hold:*

- (i)  $\mathcal{A}_{\sigma,i} \in \alpha(\mathcal{A})$ .
- (ii) Let  $m \in \mathbb{N}$ . Let  $w \in \sigma_1 \dots \sigma_m \in \Sigma^m$  such that  $\sigma_j = \sigma$  for a  $j \in \{1, \dots, m\}$  with  $j \geq i$  and  $m \geq j + (n-i)$ . Then  $w$  is rejected by  $\mathcal{A}_{\sigma,i}$ . ◻



■ **Figure 1** DFA  $\mathcal{A}_{\underline{i}}$  for  $\underline{i} \in I_m$  with  $m \in \{1, \dots, n - 1\}$  and DFA  $\mathcal{A}_{\sigma, i}$  for  $\sigma \in \Sigma, i \in \{1, \dots, n\}$ .

Now consider a word  $w = \sigma_1 \dots \sigma_m \in \Sigma^m$  with  $m \geq n + (n - 1)$  and  $\sigma_1 \dots \sigma_n \in L$ . Note that Lemma 3.4 implies  $w \notin \mathcal{L}(\mathcal{A}_{\sigma_n, i})$  where  $i \in \{1, \dots, n\}$  with  $\sigma_n \notin \Sigma_{i-1, i}$ . With this limitation of length, we only need DFAs to reject the extensions of words  $u \in L, |u| = n$  with a maximum length of  $n + (n - 2)$ . Consider such an extension  $w = \sigma_1 \dots \sigma_m \in \Sigma^m$ . That is,  $n + 1 \leq m \leq n + (n - 2)$  and  $\sigma_1 \dots \sigma_n \in L$ . This implies  $\sigma_i \in \Sigma_{i-1, i}$  for each  $i \in \{1, \dots, n\}$  but provides no information about the  $\sigma_i$  with  $i \in \{n + 1, \dots, m\}$ . Therefore, we construct DFAs rejecting every such extension not confirming to a certain structure. This structure will be key to the further DFA constructions.

For a word  $w \in \Sigma^*$ , let  $\mathcal{A}_w^!$  be the DFA rejecting exactly the words containing  $w$  as a subsequence. Clearly, the following holds:

► **Lemma 3.5.** *Let  $w \notin L, |w| = n$ . Then  $\mathcal{A}_w^! \in \alpha(\mathcal{A})$  holds.* ┘

With the DFAs  $\mathcal{A}_w^!$  for every  $w \notin L, |w| = n$  in hand, we only have to consider extensions of words  $u \in L, |u| = n$  with a maximum length of  $n + (n - 2)$  for which every subsequence of length  $n$  is in  $L$ .

Let  $w = \sigma_1 \dots \sigma_m$  be an extension satisfying these conditions. We construct a DFA  $\tilde{\mathcal{A}}_w \in \alpha(\mathcal{A})$  rejecting  $w$ . We utilize the rejecting state  $q_d$  and define  $\tilde{\mathcal{A}}_w = (\tilde{Q}_w, \Sigma, q_0, \tilde{\delta}_w, \tilde{F}_w)$  with  $\tilde{Q}_w = \{q_0, \dots, q_n\}$ ,  $\tilde{F}_w = \tilde{Q}_w \setminus \{q_d\}$  and  $\tilde{\delta}_w(q_0, w) = q_d$ . Further, we have  $\tilde{\delta}_w(q_0, v) = q_d$  for a  $v \in \Sigma^*$  only if  $\delta(q_0, v) \in \{q_d, q_{n+1}\}$ , ensuring  $\tilde{\mathcal{A}}_w \in \alpha(\mathcal{A})$ . In order to utilize  $q_d$  in this manner, the DFA  $\tilde{\mathcal{A}}_w$  simulates the behavior of  $\mathcal{A}$  for the states  $q_0, \dots, q_{d-1}$ . The task then is to select the transitions of states  $q_d, \dots, q_n$ .

If  $|\sigma_{d+1} \dots \sigma_m|_{\sigma_m} \leq n - d$  the DFA  $\tilde{\mathcal{A}}_w$  can simply advance for occurrences of  $\sigma_m$  and the first  $n - d - |\sigma_{d+1} \dots \sigma_{m-1}|_{\sigma_m}$  occurrences of letters unequal to  $\sigma_m$ . Thus, we only have to consider the case  $|\sigma_{d+1} \dots \sigma_m|_{\sigma_m} > n - d$ .

If  $\sigma_{n+1} \neq \sigma_m$  the DFA  $\tilde{\mathcal{A}}_w$  can advance for each letter in  $\Sigma$ , ensuring  $\tilde{\delta}_w(q_d, \sigma_{d+1} \dots \sigma_n) = q_n$ . Further, we can define  $\tilde{\delta}_w(q_n, \sigma_{n+1}) = q_{n-[(m-1)-(n+2)+1]}$  and  $\tilde{\delta}_w(q_n, \sigma_m) = q_d$ . Note that  $|\sigma_{n+2} \dots \sigma_{m-1}| = (m-1) - (n+2) + 1$ . Since every subsequence of  $w$  of length  $n$  is in  $L$ , we have  $\tilde{\delta}_w(q_{n-[(m-1)-(n+2)+1]}, \sigma_{n+2} \dots \sigma_{m-1}) = q_n$ .

The case  $\sigma_{n+1} = \sigma_m$  is more complex and needs a further case distinction, but the idea used above of circling back after reading an appropriate prefix can be employed again.

Lemma 3.6 summarizes these ideas:

► **Lemma 3.6.** *Let  $w \in \Sigma^*$  with  $|w| > n$  such that  $w \in \mathcal{L}(\mathcal{A}_v^!)$  for each  $v \notin L$ ,  $|v| = n$  and  $w \in \bigcap_{\sigma \in \Sigma} \mathcal{L}(\mathcal{A}_{\sigma, i_\sigma})$ , where for each  $\sigma \in \Sigma$  it is  $i_\sigma = \max(\{i \in \{1, \dots, n\} \mid \sigma \notin \Sigma_{i-1, i}\})$ . Then there exists a DFA  $\tilde{\mathcal{A}}_w \in \alpha(\mathcal{A})$  rejecting  $w$ . ◻*

Lemmas 3.3–3.6 imply Claim 3.2 (c). To be more precise, we have  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_0) \cap \mathcal{L}(\tilde{\mathcal{A}}_d) \cap \bigcap_{m=1}^{n-1} \bigcap_{i \in I_m} \mathcal{L}(\mathcal{A}_i) \cap \bigcap_{\sigma \in \Sigma} \mathcal{L}(\mathcal{A}_{\sigma, i_\sigma}) \cap \bigcap_{w \in X^!} \mathcal{L}(\mathcal{A}_w^!) \cap \bigcap_{w \in \tilde{X}} \mathcal{L}(\tilde{\mathcal{A}}_w)$ , where  $X^! = \{w \in \Sigma^n \mid w \notin L\}$  and  $\tilde{X}$  is the set of all extensions  $w$  of words  $u \in L$ ,  $|u| = n$  with  $|w| \leq n + (n-2)$  for which every subsequence of length  $n$  is in  $L$ . This proves the compositionality of  $\mathcal{A}$  and thereby Claim 3.2 (c).

### 3.2 Linear safety ADFAs

Next, we consider Claim 3.2 (d) and (e). For (d) we argue that  $\mathcal{A}$  is composite if it has the CEP, even if  $\mathcal{A}$  is a safety DFA, which makes circling back impossible. For (e) we argue that  $\mathcal{A}$  is prime if it is a safety DFA and it does not have the CEP.

First, we consider (d). We assume that  $\mathcal{A}$  has the CEP and argue that this implies compositionality. Note that we can reuse the DFAs  $\mathcal{A}_0$  and  $\mathcal{A}_i$ , while  $\tilde{\mathcal{A}}_d$  is not needed. This again leaves the task of rejecting the extensions of words  $w \in L$ ,  $|w| = n$ . But, since for every such word  $w = \sigma_1 \dots \sigma_n$  there now exist  $i \in \{0, \dots, n-2\}$ ,  $l \in \{2, \dots, n-i\}$  such that  $\delta(q_0, \sigma_1 \dots \sigma_i \sigma_{i+l} \dots \sigma_n) \in \{q_n, q_{n+1}\}$ , we can construct a DFA  $\mathcal{A}_{i,l} \in \alpha(\mathcal{A})$  rejecting every extension of  $w$ .

The DFA  $\mathcal{A}_{i,l}$  possesses states  $q_0, \dots, q_{i+l-2}, q_{i+l}, \dots, q_{n+1}$ . It simulates the behavior of  $\mathcal{A}$  for states  $q_0, \dots, q_{i-1}$ , redirecting transitions  $q_j \rightarrow q_{i+l-1}$  to  $q_i$ . From  $q_i$  it directly advances to  $q_{i+l}$  if a letter in  $\bigcup_{j=i+l}^{n+1} \Sigma_{i,j}$  is read, otherwise it advances to  $q_{i+1}$ . The states  $q_i, \dots, q_{i+l-2}$  form a loop. For states  $q_{i+l}, \dots, q_n$ , every transition leads to the direct successor state. The state  $q_{n+1}$  is a rejecting sink.

It is shown in the appendix that every extension of  $w$  is rejected by  $\mathcal{A}_{i,l}$ , where  $i$  is the largest possible value belonging to  $w$ , and that  $\mathcal{A}_{i,l} \in \alpha(\mathcal{A})$ . Thus,  $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_0) \cap \bigcap_{m=1}^{n-1} \bigcap_{i \in I_m} \mathcal{L}(\mathcal{A}_i) \cap \bigcap_{i=0}^{n-2} \bigcap_{l=2}^{n-i} \mathcal{A}_{i,l}$  holds, proving the compositionality of  $\mathcal{A}$  and thus (d).

Next, we consider (e) and assume that  $\mathcal{A}$  is a safety DFA and does not have the CEP. Thus, there is a  $w = \sigma_1 \dots \sigma_n$  such that  $\delta(q_0, \sigma_1 \dots \sigma_i \sigma_{i+l} \dots \sigma_n) \notin \{q_n, q_{n+1}\}$  holds for every  $i \in \{0, \dots, n-2\}$ ,  $l \in \{2, \dots, n-i\}$ . This implies the existence of a letter  $\sigma \in \Sigma_{n-1, n}$  with  $\sigma \notin \Sigma_{j, n+1}$  for every  $j \in \{0, \dots, n-1\}$ . We show in the appendix that  $w\sigma$  is a primality witness of  $\mathcal{A}$ , thus proving the primality of  $\mathcal{A}$  and thereby (e).

This completes our discussion of Claim 3.2 (a)–(e). Since they imply Theorem 3.1, we have characterized the compositionality of ADFAs and thereby of finite languages.

## 4 Complexity of Prime-DFA<sub>fin</sub>

After characterizing the compositionality of ADFAs and thereby of finite languages in Section 3, we now analyze the complexity of PRIME-DFA<sub>fin</sub>. We argue:

► **Theorem 4.1.** *The problem PRIME-DFA<sub>fin</sub> is NL-complete. The NL-completeness holds true even when restricting PRIME-DFA<sub>fin</sub> to DFAs with at most two letters. ◻*

■ **Algorithm 1** NL-algorithm for PRIME-DFA<sub>fin</sub>.

**Require:** DFA  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  with  $Q = \{q_0, \dots, q_m\}$  recognizing a finite language  $L$ .

**Ensure:** The DFA  $\mathcal{A}$  is prime.

```

1: Accept if  $L = \emptyset$ .
2:  $c \leftarrow 0$ 
3: for all  $i \in \{0, \dots, m\}$  do
4:   if  $q_i$  is unreachable then
5:      $c \leftarrow c + 1$ 
6:   else
7:      $j \leftarrow 0, b \leftarrow true$ 
8:     while  $j \leq i - 1$  and  $b$  do
9:       if  $q_j$  is reachable and  $\mathcal{L}(\mathcal{A}^{q_i}) = \mathcal{L}(\mathcal{A}^{q_j})$  then
10:         $c \leftarrow c + 1$ 
11:         $b \leftarrow false$ 
12:       end if
13:        $j \leftarrow j + 1$ 
14:     end while
15:   end if
16: end for
17:  $n \leftarrow (m + 1) - c - 2$ 
18: Choose nondeterministically a word  $w \in \Sigma^n$ . Reject if  $w \notin L$ .
19: Choose nondeterministically a letter  $\sigma \in \Sigma$ . Accept if  $\sigma^n \in L$ .
20: for all  $i \in \{0, \dots, m\}$  where  $q_i$  is not unreachable do
21:   Reject if  $q_i \notin F$  and  $\mathcal{L}(\mathcal{A}^{q_i}) \neq \emptyset$ .
22: end for
23: for all  $x \in \{1, \dots, n\}$  do
24:   Choose nondeterministically a word  $w = \sigma_1 \dots \sigma_n \in \Sigma^n$ . Reject if  $w \notin L$ .
25:   for all  $i \in \{0, \dots, n - 2\}, l \in \{2, \dots, n - i\}$  with  $i + l = x$  do
26:     Choose nondeterministically a word  $w' = \sigma'_1 \dots \sigma'_n \in \Sigma^n$  with  $\sigma'_{i+l} = \sigma_x$  and a word
        $v \in \Sigma^+$ . Reject if  $w' \notin L$  or if  $\sigma'_1 \dots \sigma'_i \sigma'_{i+l} \dots \sigma'_n v \notin L$ .
27:   end for
28: end for
29: Accept.

```

We begin by arguing that PRIME-DFA<sub>fin</sub> is in NL, providing an NL-algorithm for PRIME-DFA<sub>fin</sub> with Algorithm 1. The algorithm accepts in line 1 if the given DFA  $\mathcal{A}$  recognizes the empty language. Then lines 2-18 ensure that the minimal DFA belonging to  $\mathcal{A}$  is linear. Lines 19-22 ensure that  $\mathcal{A}$  is accepted if a letter  $\sigma \in \Sigma$  with  $\sigma^n \in L$  exists or else that  $\mathcal{A}$  is rejected if it is not a safety DFA. Finally, in lines 23-29 the CEP is checked for  $\mathcal{A}$ .

The NL-hardness of PRIME-DFA<sub>fin</sub> can be proven by L-reducing STCONDAG to PRIME-DFA<sub>fin</sub>, where STCONDAG is the restriction of STCON to acyclic graphs. The L-reduction is similar to the L-reduction of STCON to the emptiness problem for DFAs.

## 5 Finite Languages under Different Notions of Compositionality

So far, we have only considered  $\cap$ -compositionality. Now we will define two further notions of compositionality and characterize the compositionality of finite languages for these notions.

► **Definition 5.1.** For  $k \in \mathbb{N}_{\geq 1}$ , a DFA  $\mathcal{A}$  is  $k$ - $\cup$ -decomposable ( $k$ -DNF-decomposable) if there exist DFAs  $\mathcal{A}_1, \dots, \mathcal{A}_t$  ( $\mathcal{A}_{1,1}, \dots, \mathcal{A}_{1,t_1}, \dots, \mathcal{A}_{s,1}, \dots, \mathcal{A}_{s,t_s}$ ) with  $\mathcal{L}(\mathcal{A}) = \bigcup_{i=1}^t \mathcal{L}(\mathcal{A}_i)$  ( $\mathcal{L}(\mathcal{A}) = \bigcup_{i=1}^s \bigcap_{j=1}^{t_i} \mathcal{L}(\mathcal{A}_{i,j})$ ) and  $|\mathcal{A}_i| < k$  for every  $i$  ( $|\mathcal{A}_{i,j}| < k$  for every pair  $i, j$ ). The further concepts introduced in Definition 2.1 are defined analogously. ◻

In [10], it is correctly remarked that many results for  $\cap$ -compositionality can be trivially transferred to  $\cup$ -compositionality. For example, the complexity boundaries for PRIME-DFA established in [10] also hold for  $\cup$ -compositionality. This does not hold true for results concerning language fragments that are not closed under complement. In particular, the complement language of a finite language is not finite, but co-finite. Thus, characterizing the  $\cup$ -compositionality of finite languages is equivalent to characterizing  $\cap$ -compositionality of co-finite languages.

Also in [10], the notion of compositionality allowing both union and intersection is suggested. Note that DNF-compositionality enforces a structure similar to a disjunctive normal form, but is as strong as unrestricted union-intersection compositionality. It is correctly remarked in [10] that union-intersection compositionality - and thus, DNF-compositionality - is strictly stronger than  $\cap$ -compositionality. Obviously, it is also strictly stronger than  $\cup$ -compositionality. It is less obvious whether languages exist that are DNF-composite, but are neither  $\cap$ - nor  $\cup$ -composite. We will see that there are finite languages witnessing this.

The following result characterizes the  $\cup$ - and DNF-compositionality of finite languages:

► **Theorem 5.2.** Consider a minimal A DFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  recognizing a non-empty language. Let  $n \in \mathbb{N}$  be the length of the longest word in  $\mathcal{L}(\mathcal{A})$ . The following assertions hold:

- (i)  $\mathcal{A}$  is  $\cup$ -prime iff  $\mathcal{A}$  is linear.
- (ii)  $\mathcal{A}$  is DNF-prime iff  $\mathcal{A}$  is linear and there exists a  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$ . ◻

These conditions are similar to the conditions in Theorem 3.1, but much simpler. Let  $\mathcal{A}$  and  $n$  be as required. It is easy to show  $\cup$ - and DNF-compositionality if  $\mathcal{A}$  is not linear.

The proof of  $\cup$ -primality if  $\mathcal{A}$  is linear relies on the observation that every minimal DFA  $\mathcal{B}$  with  $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$  and  $\text{ind}(\mathcal{B}) < \text{ind}(\mathcal{A})$  has to have a rejecting sink. From this follows that no such DFA  $\mathcal{B}$  can accept a word  $w \in \mathcal{L}(\mathcal{A})$ ,  $|w| = n$ . Thus,  $\mathcal{A}$  is  $\cup$ -prime.

If  $\mathcal{A}$  is linear and there exists no  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  the DNF-compositionality of  $\mathcal{A}$  follows from [10, Example 3.2]. On the other hand, if  $\mathcal{A}$  is linear and there exists a  $\sigma \in \Sigma$  with  $\sigma^n \in \mathcal{L}(\mathcal{A})$  DNF-primality can be shown by adapting the proof of Claim 3.2 (b).

As mentioned, Theorems 3.1 and 5.2 immediately imply:

► **Theorem 5.3.** There exists a finite language that is DNF-composite but  $\cap$ - and  $\cup$ -prime. ◻

To summarize, Theorems 3.1 and 5.2 characterize the  $\cap$ -,  $\cup$ - and DNF-compositionality of ADFAs and thus of finite languages. Obviously, this characterizes the  $\cap$ -,  $\cup$ - and DNF-compositionality of co-finite languages as well. The results further imply the existence of languages that are DNF-composite but  $\cap$ - and  $\cup$ -prime.

## 6 2Minimal-DFA and S-Prime-DFA

We defined compositionality using the index of the given DFA. Thus, the compositionality of a DFA  $\mathcal{A}$  is a characteristic of  $\mathcal{L}(\mathcal{A})$ . Slightly changing the definition, using the size instead of the index, turns compositionality of  $\mathcal{A}$  into a characteristic of  $\mathcal{A}$  itself. It is interesting to analyze the effects of this change, which results in the notion of S-compositionality.

Many results known for compositionality hold for S-compositionality as well. The characterization of finite languages in Section 3 and other results concerning language fragments [10, 8, 9] are valid with only minor technical modifications. In fact, [8, 9]

already implicitly used S-compositionality instead of compositionality without discussing the differences. The upper complexity boundary of PRIME-DFA holds for S-PRIME-DFA as well. But the known lower boundary, the NL-hardness of PRIME-DFA, cannot simply be adapted for S-PRIME-DFA. The lower boundary for S-PRIME-DFA is connected to MINIMAL-DFA, since non-minimal DFAs are trivially S-composite. Note that PRIME-DFA is connected to the emptiness problem for DFAs in a similar manner [10].

We begin by discussing MINIMAL-DFA, proving the NL-hardness of 2MINIMAL-DFA. Then we formally introduce S-compositionality and prove the NL-hardness of the restriction 2S-PRIME-DFA and thereby of S-PRIME-DFA as well. We also prove the NL-hardness of the restriction 2PRIME-DFA, so far only known for the unrestricted problem PRIME-DFA.

## 6.1 NL-hardness of 2Minimal-DFA

As mentioned, the NL-hardness and thus NL-completeness of  $k$ MINIMAL-DFA for  $k \in \mathbb{N}_{\geq 3}$  is folklore, while the NL-hardness of 2MINIMAL-DFA appears to be open. We prove:

► **Theorem 6.1.** *The problem 2MINIMAL-DFA is NL-hard and thus NL-complete.* ◻

The NL-hardness of 3MINIMAL-DFA can be proven by L-reducing 2STCON to 3MINIMAL-DFA. This known reduction uses an additional letter and cannot be used to prove the NL-hardness of 2MINIMAL-DFA. We give an L-reduction of 2STCON not using an additional letter, proving the NL-hardness and thus the NL-completeness of 2MINIMAL-DFA.

Let  $(G, s, t)$  be an input for 2STCON. That is,  $G = (V, E)$  is a graph with a maximum outdegree of two and  $s, t \in V$  are nodes of  $G$ . We construct a DFA  $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$  with  $\Sigma = \{0, 1\}$ , which is minimal iff there exists a path in  $G$  from  $s$  to  $t$ . If  $s = t$  such a path exists trivially and we can construct the minimal DFA for the empty language. Thus, we only have to consider the case  $s \neq t$ . W.l.o.g. we assume  $V = \{0, \dots, n-1\}$  and  $s = 0, t = n-1$ .

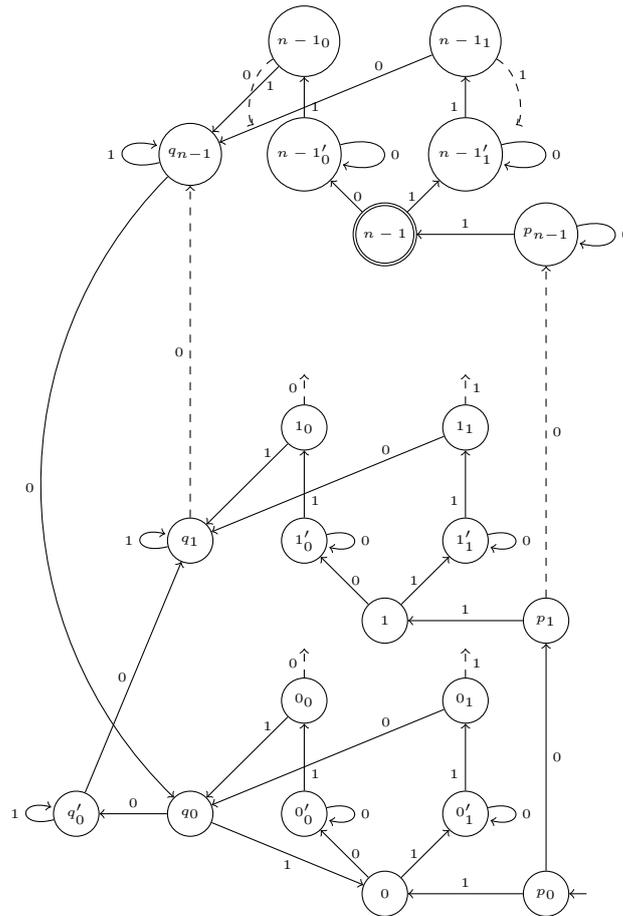
Let  $\mathcal{A}' = (Q', \Sigma, 0, \delta', F')$  be the DFA constructed out of  $G$  in the usual manner, that is, by turning nodes into states, edges into transitions, setting the state 0 as the initial state and  $n-1$  as the only accepting state. For  $\mathcal{A}$ , we introduce the new states  $p_0, \dots, p_{n-1}$ , called  $p$ -states, the new states  $q_0, \dots, q_{n-1}$  and  $q'_0$ , called  $q$ -states, and for each  $i \in Q'$  the states  $i'_0, i'_1, i_0, i_1$ . We call the states  $i, i'_0, i'_1, i_0, i_1$  for  $i \in Q'$   $v$ -states. We say that states  $p_i, q_i, i, i'_0, i'_1, i_0, i_1$  for an  $i \in Q'$  are located on the same layer. Figure 2 specifies the DFA  $\mathcal{A}$  constructed for the L-reduction. We now discuss the key ideas of this construction.

First, note that the idea of the  $p$ - and  $q$ -states is similar to the known L-reduction of 2STCON to 3MINIMAL-DFA. The  $p$ -states are used to access every state in  $Q$ , thus avoiding unreachable states. The  $q$ -states are used to allow the return to 0 from every state.

Second, we cannot use an additional letter to switch from  $p_i$  to  $i$  to  $q_i$ . Thus, letter 1 is used to leave the  $p$ -states and to exit  $q_0$  to state 0. Letter 0 is used to advance to the next layer in both the  $p$ - and  $q$ -states. To allow switching from the  $v$ -states to the  $q$ -states, we introduce for each  $i \in Q'$  a component consisting of  $i$  and the two branches  $i'_0, i_0$  and  $i'_1, i_1$ . The states  $i'_0, i'_1$  are waiting states used to prove the non-equivalence of  $q$ - and  $v$ -states. The states  $i_0, i_1$  implement on the one hand the original transitions in  $\mathcal{A}'$ , that is,  $\delta(i_j, j) = \delta'(i, j)$ , and on the other hand the transitions into the  $q$ -states, that is,  $\delta(i_j, 1-j) = q_i$ .

Third, an extra  $q$ -state  $q'_0$  is introduced, which is only directly accessible from  $q_0$ . Without  $q'_0$  the situation  $\delta(1_1, 1) = 0 = \delta(q_0, 1)$  and  $\delta(1_1, 0) = q_1 = \delta(q_0, 0)$  would be possible, immediately implying the non-minimality of  $\mathcal{A}$ . The introduction of  $q'_0$  solves this problem.

Note that there is a path from 0 to  $n-1$  in  $\mathcal{A}$  iff there is such a path in  $G$ . Using this it follows that  $\mathcal{A}$  is minimal iff there exists a path from 0 to  $n-1$  in  $G$ . Since  $\mathcal{A}$  can obviously be constructed in logarithmic space, the given construction is indeed an L-reduction of 2STCON to 2MINIMAL-DFA. Consequently, 2MINIMAL-DFA is NL-hard.



■ **Figure 2** DFA  $\mathcal{A}$  constructed for the L-reduction of 2STCON to 2MINIMAL-DFA. The  $j$ -transitions exiting states of the form  $i_j$  are only indicated.

### 6.2 Complexity of S-Prime-DFA

We end our discussion by using the construction presented in Section 6.1 to establish complexity boundaries for S-PRIME-DFA. First, we define the notion of S-compositionality.

► **Definition 6.2.** A DFA  $\mathcal{A}$  is *S-composite* if there is a  $k \in \mathbb{N}_{\geq 1}, k < |\mathcal{A}|$  such that  $\mathcal{A}$  is  $k$ -decomposable. Otherwise,  $\mathcal{A}$  is *S-prime*. ◻

We denote the problem of deciding S-primality for a given DFA with S-PRIME-DFA and the restriction of S-PRIME-DFA to DFAs with at most  $k \in \mathbb{N}_{\geq 2}$  letters with  $k$ S-PRIME-DFA.

Note that the proof used in [10] to show that PRIME-DFA is in EXPSPACE is applicable for S-PRIME-DFA with only slight modifications. Next, note that the L-reduction of the emptiness problem for DFAs to PRIME-DFA used in [10] to prove the NL-hardness of PRIME-DFA relies on the fact that every DFA recognizing the empty language is prime. Thus, it is not easily adaptable for S-PRIME-DFA. Instead, the NL-hardness of 2S-PRIME-DFA is shown by using a reduction from 2STCON, which adapts the construction outlined in Section 6.1. We get:

► **Theorem 6.3.** *The problems S-PRIME-DFA and kS-PRIME-DFA for  $k \in \mathbb{N}_{\geq 2}$  are in EXPSPACE and they are NL-hard.* ◻

Further, we denote with  $k$ PRIME-DFA the restriction of PRIME-DFA to DFAs with at most  $k \in \mathbb{N}_{\geq 2}$  letters and remark that the results presented in [10] can be expanded to:

► **Theorem 6.4.** *The problems PRIME-DFA and  $k$ PRIME-DFA for  $k \in \mathbb{N}_{\geq 2}$  are in EXPSPACE and they are NL-hard.* ─

This ends our discussion of the complexity of S-PRIME-DFA and its restrictions, in which we have applied the construction outlined in Section 6.1 to prove NL-hardness.

## 7 Discussion

We studied the intersection compositionality, also denoted with  $\cap$ -compositionality, of regular languages. We added to the existing line of research focusing on fragments of the regular languages by analyzing the  $\cap$ -compositionality of ADFAs and thereby of finite languages. This research was in part motivated by existing results concerning the concatenation compositionality of finite languages.

We completely characterized the  $\cap$ -compositionality of ADFAs and thus finite languages. Using this characterization we proved the NL-completeness of PRIME-DFA<sub>fin</sub>. Thus, finite languages are significantly easier to handle under  $\cap$ -compositionality than under concatenation compositionality, where the respective primality problem for finite languages is NP-hard [15].

With notions of compositionality using union and both union and intersection already suggested in [10], we formally introduced the notions of  $\cup$ - and DNF-compositionality. We characterized the  $\cup$ - and DNF-compositionality of finite languages, which proved to be far simpler than the characterization of  $\cap$ -compositionality. These results also imply the characterization of the  $\cap$ -,  $\cup$ - and DNF-compositionality of co-finite languages.

This suggests that the key feature of finite languages regarding compositionality is not the finiteness of the languages per se, but rather the existence of only finitely many meaningfully different runs of the respective DFAs, a feature finite languages have in common not only with co-finite languages, but also with languages whose minimal DFAs allow for cycles in both accepting and rejecting sinks. A logical next step would therefore be the characterization of the compositionality of these DFAs.

We also note that in our proofs we employed  $\cap$ -compositionality results concerning a different language fragment, namely co-safety DFAs, studied in [10]. This suggests the possibility of employing the results concerning finite languages in future analyses and stresses the usefulness of working with language fragments. We provided one application of the results concerning finite languages by using them to prove the existence of a language that is DNF-composite but  $\cap$ - and  $\cup$ -prime.

Furthermore, we presented a proof of the NL-hardness and thereby NL-completeness of the basic problem 2MINIMAL-DFA. While the NL-hardness of  $k$ MINIMAL-DFA for  $k \in \mathbb{N}_{\geq 3}$  is folklore, this result appears to be new.

We utilized this result to establish the known complexity boundaries of PRIME-DFA for the here newly introduced problem S-PRIME-DFA. We extended these results to the restrictions  $k$ PRIME-DFA and  $k$ S-PRIME-DFA for  $k \in \mathbb{N}_{\geq 2}$ .

While it is interesting that a slight variation in the definition of  $\cap$ -compositionality, which does not touch the validity of most results, requires a whole new approach to establish the known lower complexity boundary, the big task of closing the doubly exponential complexity gap for PRIME-DFA still remains. And now, this gap exists for S-PRIME-DFA as well.

Therefore, with the analysis of language fragments, further notions of compositionality, and the complexity gaps for PRIME-DFA and S-PRIME-DFA, there is still need for further research.

## References

- 1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. URL: <https://mitpress.mit.edu/9780262026499/principles-of-model-checking/>.
- 2 Sang Cho and Dung T. Huynh. The parallel complexity of finite-state automata problems. *Inf. Comput.*, 97(1):1–22, 1992. doi:10.1016/0890-5401(92)90002-W.
- 3 Willem P. de Roever, Hans Langmaack, and Amir Pnueli, editors. *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*. Springer, 1998. doi:10.1007/3-540-49213-5.
- 4 Henning Fernau and Markus Holzer. Personal communication.
- 5 Peter Gazi and Branislav Rován. Assisted problem solving and decompositions of finite automata. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrát, and Mária Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, volume 4910 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2008. doi:10.1007/978-3-540-77566-9\_25.
- 6 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.
- 7 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 8 Ismaël Jecker, Orna Kupferman, and Nicolas Mazzocchi. Unary prime languages. 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 51:1–51:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.51.
- 9 Ismaël Jecker, Nicolas Mazzocchi, and Petra Wolf. Decomposing permutation automata. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.18.
- 10 Orna Kupferman and Jonathan Mosheiff. Prime languages. *Inf. Comput.*, 240:90–107, 2015. doi:10.1016/j.ic.2014.09.010.
- 11 Niklas Lauffer, Beyazit Yalcinkaya, Marcell Vazquez-Chanlatte, Ameesh Shah, and Sanjit A. Seshia. Learning deterministic finite automata decompositions from examples and demonstrations. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 1–6. IEEE, 2022. doi:10.34727/2022/isbn.978-3-85448-053-2\_39.
- 12 Wim Martens, Matthias Niewerth, and Thomas Schwentick. Schema design for XML repositories: complexity and tractability. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 239–250. ACM, 2010. doi:10.1145/1807085.1807117.
- 13 Alexandru Mateescu, Arto Salomaa, and Sheng Yu. Factorizations of languages and commutativity conditions. *Acta Cybern.*, 15(3):339–351, 2002. URL: <https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3583>.
- 14 Arto Salomaa and Sheng Yu. On the decomposition of finite languages. In Grzegorz Rozenberg and Wolfgang Thomas, editors, *Developments in Language Theory, Foundations, Applications, and Perspectives, Aachen, Germany, 6-9 July 1999*, pages 22–31. World Scientific, 1999. doi:10.1142/9789812792464\_0003.
- 15 Philip Sieder. A lower bound for primality of finite languages. *CoRR*, abs/1902.06253, 2019. arXiv:1902.06253.

## 83:14 Decomposing Finite Languages

- 16 Stavros Tripakis. Compositionality in the science of system design. *Proc. IEEE*, 104(5):960–972, 2016. doi:10.1109/JPROC.2015.2510366.
- 17 Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 332–344. IEEE Computer Society, 1986. URL: <https://hdl.handle.net/2268/116609>.
- 18 Wojciech Wieczorek. An algorithm for the decomposition of finite languages. *Log. J. IGPL*, 18(3):355–366, 2010. doi:10.1093/jigpal/jzp032.

# Dependent $k$ -Set Packing on Polynomoids

Meng-Tsung Tsai ✉ 

Institute of Information Science, Academia Sinica, Taipei, Taiwan

Shi-Chun Tsai ✉ 

Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

Tsung-Ta Wu ✉

Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

---

## Abstract

Specialized hereditary systems, e.g., matroids, are known to have many applications in algorithm design. We define a new notion called  *$d$ -polynomoid* as a hereditary system  $(E, \mathcal{F} \subseteq 2^E)$  so that every two maximal sets in  $\mathcal{F}$  have less than  $d$  elements in common. We study the problem that, given a  $d$ -polynomoid  $(E, \mathcal{F})$ , asks if the ground set  $E$  contains  $\ell$  disjoint  $k$ -subsets that are not in  $\mathcal{F}$ , and obtain a complexity trichotomy result for all pairs of  $k \geq 1$  and  $d \geq 0$ . Our algorithmic result yields a sufficient and necessary condition that decides whether each hypergraph in some classes of  $r$ -uniform hypergraphs has a perfect matching, which has a number of algorithmic applications.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Matroids and greedoids

**Keywords and phrases** Hereditary Systems, Hypergraph Matchings, Complexity Trichotomy

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.84

**Funding** This research was supported in part by the National Science and Technology Council under contract NSTC grants 108-2221-E-009-051-MY3, 109-2221-E-001-025-MY3, and 112-2221-E-001-007.

**Acknowledgements** We sincerely thank anonymous reviewers for their helpful comments.

## 1 Introduction

Finding the conditions that decide whether an  $r$ -uniform hypergraph  $H$  contains a perfect matching has received much attention. Some conditions are based on the minimum degree of a vertex in  $H$  [14, 34, 35], and some are based on the minimum degree of a set of  $r - 1$  vertices in  $H$  [38]. More conditions are known for bipartite hypergraphs, such as Hall's [24], Aharoni's [1, 3], and Haxell's [27], and multipartite hypergraphs [11, 2]. Because finding a maximum matching for  $r$ -uniform hypergraphs with  $r \geq 3$  is APX-complete [32, 29], any computationally efficient conditions to decide whether an  $r$ -uniform hypergraph with  $r \geq 3$  contains a perfect matching cannot be both sufficient and necessary unless  $P = NP$ . Indeed, all the conditions above except Hall's are sufficient but not necessary. In the literature, a number of polynomial-time algorithms to compute perfect matchings for dense  $r$ -uniform hypergraphs are known [33, 26, 25].

In this paper, we give a sufficient and necessary condition that, for any pair of integers  $k > d \geq 0$ , the  $k$ -uniform hypergraph

$$H = \left( V, E = \left\{ e \in \binom{V}{k} : e \not\subseteq S_i \text{ for all } i \in [m] \right\} \right)$$

has a perfect matching, where  $\binom{V}{k}$  denotes the collection of all  $k$ -subsets of  $V$  and  $S_1, S_2, \dots, S_m$  are subsets of  $V$  with  $|S_i \cap S_j| < d$  for all  $i \neq j \in [m] := \{1, 2, \dots, m\}$ . We prove also the hardness of finding a maximum matching for such hypergraphs when  $k \leq d$ . Combining the above, we obtain a complexity trichotomy for our problem for all pairs of  $k$  and  $d$ , detailed in Theorem 2.



© Meng-Tsung Tsai, Shi-Chun Tsai, and Tsung-Ta Wu;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 84; pp. 84:1–84:15



Leibniz International Proceedings in Informatics

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

To better understand the structure of the hypergraphs defined above and compare our results with related work, we restate finding a maximum matching for such hypergraphs as the dependent  $k$ -set packing on a kind of hereditary systems that we call **polynomoids**, defined in Definition 1. Our notation for hereditary systems follows West [42].

► **Definition 1** ( $d$ -Polynomoid). *Let  $d \geq 0$  be an integer. Let  $(E, \mathcal{F})$  be a tuple where  $E$  is a finite set and  $\mathcal{F} \subseteq 2^E$  is a non-empty collection of some subsets of  $E$ . The sets in  $\mathcal{F}$  are called **independent sets**, and the other subsets of  $E$  are **dependent sets**. We say that  $P = (E, \mathcal{F})$  is a  **$d$ -polynomoid** if  $P$  satisfies the following two properties:*

- *Hereditary Property: For every  $B \in \mathcal{F}$ , if  $A \subseteq B$  then  $A \in \mathcal{F}$ .*
- *Join Property: For every  $A, B \in \mathcal{F}$ , if  $|A \cap B| \geq d$  then  $A \cup B \in \mathcal{F}$ .*

If the join property is removed, then  $P = (E, \mathcal{F})$  is a hereditary system [42]; if the join property is replaced with the exchange property, then  $P$  is a matroid [37].

Given a  $d$ -polynomoid  $P = (E, \mathcal{F})$  and integers  $k \geq 1, \ell \geq 0$ , the **dependent  $k$ -set packing** for  $P$  asks if there exist  $\ell$  pairwise disjoint  **$k$ -sets** not in  $\mathcal{F}$ , where a set is called  $k$ -set if it consists of  $k$  elements. If the  $\ell$  disjoint  $k$ -sets exist, then output them. Our main result is a complexity trichotomy for the dependent  $k$ -set packing problem on polynomoids, stated formally in Theorem 2. An illustration is depicted in Figure 1.

► **Theorem 2.** *The time complexities of the dependent  $k$ -set packing on  $d$ -polynomoids for all pairs of integers  $k \geq 1, d \geq 0$  can be classified into the following three categories:*

1. *If  $k \leq d$  and  $k \geq 3$ , there exists a  $d$ -polynomoid  $P$  so that the dependent  $k$ -set packing for  $P$  is APX-complete.*
2. *If  $k \leq d$  and  $k \leq 2$ , then:*
  - *for  $k = 2$ , there exists a  $d$ -polynomoid  $P$  so that the dependent  $k$ -set packing for  $P$  is as hard as the matching problem on ordinary graphs (i.e. 2-uniform hypergraphs);*
  - *for  $k = 1$ , this is a degenerate case solvable in  $O(|E|q(1))$  time, where  $q$  is a function defined below.*
3. *Otherwise  $k > d$ , for any  $d$ -polynomoid  $P = (E, \mathcal{F})$ ,*

*$E$  contains  $\lfloor |E|/k \rfloor$  disjoint dependent  $k$ -sets if and only if  $r(E) \leq (1 - 1/k)|E|$ ,*

*where  $r(E)$  is the size of a maximum independent subset of  $E$ . The dependent  $k$ -set packing can be found in  $O(k|E|q(2k))$  time, where  $q(t)$ <sup>1</sup> denotes a monotone function that upper-bounds the running time to test whether a  $t$ -subset of  $E$  is independent.*

It may be worth noting that for some polynomoids computing  $r(E)$  requires quadratic time unless the 3SUM conjecture fails, as shown in Section 2.2. To obtain Theorem 2, it suffices to test  $r(E) \leq (1 - 1/k)|E|$  without computing the exact value of  $r(E)$ , which can be tested in deterministic linear time (Section 4.1). In addition, greedily grouping the elements in a largest independent set with those from other independent sets in general cannot work correctly because the condition  $r(E) \leq (1 - 1/k)|E|$  may be violated in the residual polynomoid obtained from the initial polynomoid with a greedy removal of the elements in a largest independent set and the corresponding grouped elements from other independent sets.

In addition to the complexity trichotomy result, Theorem 2 can also be used to yield a sufficient and necessary condition for each polynomoid when a perfect packing exists, such as Corollary 3 (also follows from [14]), Corollary 4, Corollary 22, and Corollary 24 (also proven in [7]).

<sup>1</sup> Indeed, it has to be written as  $q_P(t)$  because it varies among different polynomoids. We suppress the subscript  $P$  when the context is clear.

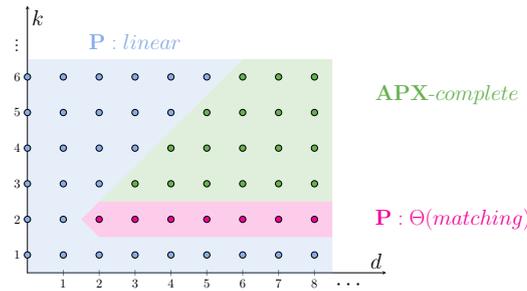


Figure 1 A complexity trichotomy for the dependent  $k$ -set packing.

► **Corollary 3.** *Let  $G$  be a 3-uniform hypergraph with at least four vertices. If the number of vertices in  $G$  is a multiple of 3 and every two hyperedges in  $G$  have at most one vertex in common, then the complement of  $G$  has a perfect matching.*

**Proof.** Let  $P = (E, \mathcal{F})$  where  $E$  is the set of vertices in  $G$  and  $\mathcal{F}$  is the set of hyperedges in  $G$ . By definition,  $P$  is a 2-polynomoid. By setting  $(k, d) = (3, 2)$  and applying Theorem 2 on  $P$ , we are done. ◀

We remark that a number of related works discuss the independent set partition for hereditary systems [20, 43], and the minimal dependent set packing [39] and partition [17, 41, 31, 13, 21] for matroids. Note that partition problems can be reduced to packing problems.

### 1.1 Example Polynomoids

There are a number of structures that satisfy the requirements of polynomoids. We list three of them below, and more can be found in Appendix B.

- Let  $E$  be a finite set of points in  $\mathbb{R}^2$  and

$$\mathcal{F} = \{E' \subseteq E : \text{all points in } E' \text{ are colinear}\}.$$

$P = (E, \mathcal{F})$  is a 2-polynomoid because:

- Hereditary Property: Let  $S$  be a point set in  $\mathbb{R}^2$ . If all points in  $S$  are colinear, then all points in any subset of  $S$  also are colinear.
- Join Property: Let  $S_1, S_2$  be two point sets in  $\mathbb{R}^2$ . If all points in each of  $S_1$  and  $S_2$  are colinear and  $|S_1 \cap S_2| \geq 2$ , then all points in  $S_1 \cup S_2$  are colinear.

The dependent  $k$ -set packing problem for  $P$  asks if  $E$  contains  $\ell$  disjoint  $k$ -sets so that the points in each  $k$ -set are not colinear. In particular, for  $k = 3$ , the points in each  $k$ -set are on a circle with a finite radius, as depicted in Figure 2a. By Theorem 2, this problem can be solved in  $O(|E|)$  time as  $k, q(2k) = O(1)$ .

► **Remark.** More generally, the above example can be generalized to any set of degree- $d$  univariate polynomials for any  $d \geq 1$ . Let  $E$  denote a finite set of distinct points in  $\mathbb{R}^2$ . We may need a rotation of axes to ensure that no two points in  $E$  have the same  $x$ -coordinate. Let  $\mathcal{L}$  denote a collection of polynomials with degree  $d$ . Let  $\mathcal{F}$  denote the collection of all the subsets  $E'$  of  $E$  that some polynomial in  $\mathcal{L}$  passes through all points in  $E'$ . It is not hard to check that such an  $(E, \mathcal{F})$  is a  $d$ -polynomoid. This motivates us to call the hereditary systems defined in Definition 1 *polynomoids*.

2. Let  $G = (V, E)$  be an undirected simple graph and

$$\mathcal{F} = \{E' \subseteq E : \text{all edges in } E' \text{ have a vertex in common}\}.$$

Hence, for any  $A \in \mathcal{F}$ , the subgraph of  $G$  induced by the edges in  $A$  is a star graph.  $P = (E, \mathcal{F})$  is a 2-polynomoid because:

- Hereditary Property: Any subgraph of a star graph also is a star.
- Join Property: If two star subgraphs have at least two edges in common, then their union also is a star.

The dependent  $k$ -set packing problem for  $P$  asks if  $E$  contains  $\ell$  disjoint  $k$ -sets so that the edges in each  $k$ -set do not form a star graph. In particular, for  $k = 3$ , the edges in each  $k$ -set edge-induce a triangle or a *linear forest*, i.e. each component in the forest is a path, as depicted in Figure 2b. Because the union of the edges in a triangle and those in any 3-edge non-star graph can be partitioned into two 3-edge linear forests (Lemma 25), the problem of partitioning the edges in a graph into 3-edge linear forests is linear-time reducible to the dependent  $k$ -set packing for  $P$ . By Theorem 2, both problems can be solved in  $O(|E|)$  time as  $k, q(2k) = O(1)$ .

► **Remark.** It is shown in [9, 6, 4] that an  $m$ -edge undirected simple graphs with maximum degree  $\Delta$ , except for a finite number of exceptions, can be edge-partitioned into  $3P_2$ s, i.e., three vertex-disjoint edges if and only if  $m$  is a multiple of 3 and  $\Delta \leq m/3$ . By Theorem 2 and the above discussion, we obtain an analogous result that:

► **Corollary 4.** *An  $m$ -edge undirected simple graph with maximum degree  $\Delta$  can be edge-partitioned into 3-edge linear forests if and only if  $m$  is a multiple of 3 and  $\Delta \leq 2m/3$ .*

Let  $\{H_1, H_2, \dots, H_t\}$ -decomposition be the problem that, given an undirected simple graph  $G = (V, E)$ , decide whether  $E$  can be partitioned into subsets so that each subset edge-induces a subgraph isomorphic to  $H_i$  for some  $i \in [t]$ . It is conjectured in [40] that  $\{H_1, H_2, \dots, H_t\}$ -decomposition is NP-complete if and only if  $\{H_i\}$ -decomposition is NP-complete for all  $i \in [t]$ . Let  $P_\ell$  be a path of  $\ell$  nodes. Let  $P_i \cup P_j$  be the union of vertex-disjoint  $P_i$  and  $P_j$  and let  $kP_i$  be the union of  $k$  vertex-disjoint  $P_i$ s. By the above conjecture and the fact that  $P_4$ -decomposition is NP-complete [28, 15] but  $P_3 \cup P_2$ -decomposition [18, 10] and  $3P_2$ -decomposition [9, 6, 4] are polynomial-time solvable, partitioning the edge set of an undirected simple graph into 3-edge linear forests shall (assuming the conjecture holds) be solvable in polynomial time. Our above linear-time algorithm gives an example that supports the conjecture.

3. Let  $G = (V, A)$  be an edge-weighted directed graph and

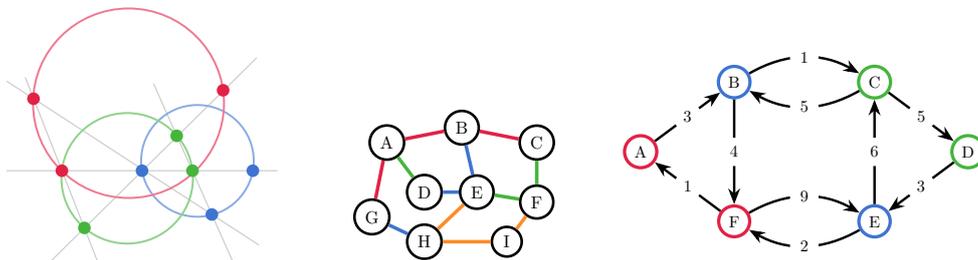
$$\mathcal{F}_\Delta = \{V' \subseteq V : \text{a minimum } st\text{-cut in } G \text{ has weight at least } \Delta \text{ for every } s \neq t \in V'\},$$

where an  $st$ -cut is a partition of  $V$  into two disjoint sets  $S$  and  $T$  with  $s \in S, t \in T$  and the weight of an  $st$ -cut is defined to be the sum of weights on the directed edges from  $S$  to  $T$ . Note that by definition  $\mathcal{F}_\Delta$  contains  $\emptyset$  and all singleton sets.  $P = (V, \mathcal{F}_\Delta)$  is a 1-polynomoid because:

- Hereditary Property: If a minimum  $st$ -cut in  $G$  has weight at least  $\Delta$  for all pairs of  $s \neq t \in V'$ , then it also applies for all pairs of vertices in a subset of  $V'$ .
- Join Property: Let  $A, B \in \mathcal{F}_\Delta$  and  $z$  be a vertex that  $A, B$  have in common. For any pair of  $s \in A, t \in B$ , a minimum  $st$ -cut  $(S, T)$  that separates  $s, t$  has either  $z \in S$  or  $z \in T$ . We assume w.l.o.g. that  $z \in S$ . Since a minimum  $tz$ -cut and a minimum  $zt$ -cut both have weights at least  $\Delta$ , the  $(S, T)$ -cut also has weight at least  $\Delta$ .

The dependent  $k$ -set packing problem for  $P$  asks if  $V$  contains  $\ell$  disjoint  $k$ -sets so that for each  $k$ -set there is a cut in  $G$  of weight less than  $\Delta$  that separates the vertices in it, as depicted in Figure 2c. By Theorem 2, this problem can be solved in  $O(k|V|k^2f(G))$  time as  $q(2k) = O(k^2f(G))$  where  $f(G)$  denotes the running time of exact maxflow computation between two distinct nodes  $s$  and  $t$  on  $G$ .

► Remark. A naive approach for this problem needs to compute the minimum  $st$ -cuts for all pairs of  $s, t \in V$ , but ours needs only  $O(k^3V) = O(V)$  pairs. For undirected graphs, by Gomory-Hu trees [23] the number of the minimum  $st$ -cuts that are needed to compute is also  $O(V)$ . For unweighted directed graphs, the running time also can be reduced by the approach in [12]. For weighted directed graphs, to the best of our knowledge, our algorithm is the first one with running time matching the current best algorithm for weighted undirected graphs.



■ **Figure 2** (a) The figure to the left: a partition of the given points into 3-sets so that the points in each 3-set are on a circle with a finite radius. (b) The figure in the middle: a partition of the edges into 3-sets so that the edges in each 3-set induces a linear forest. (c) The figure to the right: a partition of the vertices into 2-sets so that for each 2-set there is a cut in  $G$  has weight less than 7 that separates vertices in it.

## 1.2 Sharpness of Our Result

The two properties of polymoids are essential to make the dependent  $k$ -set packing for polymoids solvable in linear time for  $k > d$ .

- Case I: if the hereditary property is removed from the Definition 1, then we have an example problem for  $k > d$  that cannot be solved in polynomial time unless  $P = NP$ . Let  $G$  be an undirected graph. Let  $(E, \mathcal{F})$  be a set system where  $E$  denotes the vertex set of  $G$  and  $\mathcal{F}$  consists of all the subsets  $E'$  of  $E$  so that the subgraph of  $G$  induced by the vertices in  $E'$  are connected. Note that, for any  $E_1, E_2 \in \mathcal{F}$ , if  $|E_1 \cap E_2| \geq 1$ , then  $E_1 \cup E_2 \in \mathcal{F}$ . Thus, this set system corresponds to the case  $d = 1$ .

Set  $k = 3$ , so  $k > d$ . To find a dependent  $k$ -set packing for  $(E, \mathcal{F})$ , it is equivalent to asking whether  $E$  contains  $\ell$  disjoint  $k$ -sets so that the subgraph in  $G$  induced by the vertices in each  $k$ -subset is disconnected. For  $k = 3$ , this problem is equivalent to asking whether  $E$  contains  $\ell$  disjoint  $k$ -sets so that the subgraph in  $\bar{G}$  (the complement graph of  $G$ ) induced by the vertices in each  $k$ -set is connected, which is known to be NP-complete [16].

- Case II: if the join property is removed from the Definition 1, then the parameter  $d$  is removed. So the complexity trichotomy in Figure 1 collapses. For  $k \geq 3$ , then it is APX-complete; for  $k = 2$ , then it is as hard as matching [36]. It is not known how to solve either case in linear time.

### 1.3 Paper Organization

In Section 2, we devise a polynomial-time algorithm to find a maximum independent set for any polymoid  $P$ , assuming that the independence oracle can be queried in polynomial time, and prove that no algorithms can solve this problem in truly subquadratic time unless the 3SUM conjecture fails. Then, in Section 3, we relate the problem of finding a maximum independent set and that of finding a largest dependent  $k$ -set packing for any  $d$ -polymoids with  $k > d$ . Because finding a maximum independent set for polymoids is 3SUM-hard in general, we devise a deterministic linear-time algorithm without involving the exact computation of the maximum independent sets in Section 4. We prove the hardness for the case of  $k \leq d$  in Appendix A. Finally, we present more applications of our results in Appendix B and place omitted proofs in Appendix C.

## 2 Maximum Independent Sets

For any matroid, finding a maximum independent set can be done greedily because all maximal independent sets have equal size. Since maximal independent sets of a polymoid may have different sizes, the greedy approach for matroids cannot be applied to polymoids. In what follows, we devise a polynomial-time algorithm to find a maximum independent set for any polymoid, assuming that testing whether a set is independent can be done in polynomial time. In addition, we prove that this problem cannot be solved by any truly subquadratic-time algorithm even if the independence oracle can be decided in time linear in the input size unless the 3SUM conjecture fails.

### 2.1 A Polynomial-Time Algorithm

Our polynomial-time algorithm for finding a maximum independent set is mainly based on the following key lemma.

► **Lemma 5.** *Let  $P = (E, \mathcal{F})$  be a  $d$ -polymoid for some  $d \geq 0$ . For any  $d$ -subset  $C$  of  $E$ , precisely one of the following two statements holds:*

- *No maximal independent sets in  $P$  contain  $C$ .*
- *Exactly one maximal independent set in  $P$  contains  $C$ .*

**Proof.** If  $C$  is dependent, then no independent set contains  $C$  due to the hereditary property of  $P$ . If  $C$  is independent and there exist two distinct maximal independent sets  $M_1, M_2$  of  $P$  that contain  $C$  as a subset, then

$$|M_1 \cap M_2| \geq |C| = d.$$

By the join property of  $P$ ,  $M_1 \cup M_2$  is independent. Since  $M_1 \neq M_2$  and they are maximal, we have

$$|M_1 \cup M_2| \geq \max\{|M_1|, |M_2|\} + 1,$$

contradicting the maximality of  $M_1$  and  $M_2$ . Therefore, precisely one maximal independent set contains  $C$ . Each of the above two cases corresponds to one of the claimed statements. ◀

Lemma 5 yields an efficient algorithm that, for any independent set  $C$  of a  $d$ -polymoid with size at least  $d$ , finds “the” maximal independent set containing  $C$  as a subset. Formally, we state it in Corollary 6.

► **Corollary 6.** *Let  $P = (E, \mathcal{F})$  be a  $d$ -polymoid for some  $d \geq 0$ . For any  $C \in \mathcal{F}$  with size at least  $d$ , finding the maximal independent set  $M_C$  that contains  $C$  as a subset can be done in  $O(|E|q(d+1))$  time.*

**Proof.** By Lemma 5, there is a unique independent set  $M_C$  that contains any  $d$ -subset  $C_d$  of  $C$  as a subset. Hence, for any  $x \in E$ , if  $C_d \cup \{x\} \in \mathcal{F}$ , then  $x \in M_C$ . Testing whether  $C_d \cup \{x\} \in \mathcal{F}$  for all  $x \in E$  can be realized by invoking the independence oracle of  $P$  on  $(d+1)$ -subsets of  $E$   $O(|E|)$  times. Thus, the total running time is  $O(|E|q(d+1))$ . ◀

Lemma 5 and Corollary 6 imply that for any polymoid  $P = (E, \mathcal{F})$  finding a maximum independent set can be done in  $O(|E|^{d+1}q(d+1))$  time. This can be seen from the following two cases.

1. If the size  $s$  of maximum independent sets in  $P$  is at most  $d$ , then they can be found by invoking the independence oracle once for each subset of  $E$  that has size  $\leq d$ . Hence, the running time is

$$\sum_{i=0}^d \binom{|E|}{i} q(i) = O(|E|^d q(d)) \text{ for } |E| \geq 2 \text{ or } O(q(1)) \text{ for } |E| \leq 1.$$

2. Otherwise, there is a maximum independent set  $M$  that has size  $> d$ . Because  $|M| > d$  and the hereditary property of  $P$ ,  $M$  contains a subset  $M_d$  of size  $d$  in  $\mathcal{F}$ . By Lemma 5, exactly one maximal independent set  $W$  contains  $M_d$ , so  $W = M$ . By Corollary 6,  $W = M_d \cup \{e \in E : M_d \cup \{e\} \in \mathcal{F}\}$ , which can be found in  $O(|E|q(d+1))$  time. Hence, the total running time is at most

$$|E|q(d+1) \cdot \binom{|E|}{d} = O(|E|^{d+1}q(d+1)).$$

The implementations of the above two cases can be unified as in the following pseudocode.

■ **Algorithm 1** Finding a maximum independent set for polymoids.

---

**input** : a  $d$ -polymoid  $P = (E, \mathcal{F})$   
**output** : a maximum independent set of  $P$   
 $A \leftarrow \emptyset$ ;  
**foreach**  $S \in \{E' \subseteq E : |E'| \leq d\}$  **do**  
    **if**  $S \in \mathcal{F}$  **then**  
         $M \leftarrow$  a maximal independent set in  $\mathcal{F}$  that contains  $S$ ;  
        **if**  $|M| > |A|$  **then**  
             $A \leftarrow M$ ;  
**return**  $A$ ;

---

As a consequence, we have:

► **Theorem 7.** *For any  $d$ -polymoid  $P = (E, \mathcal{F})$ , given an independence oracle  $\mathcal{O}_{ind} : 2^E \rightarrow \{0, 1\}$  that tests whether a  $t$ -subset of  $E$  is contained in  $\mathcal{F}$  in  $q(t)$  time where  $q$  is a monotone function, then finding an independent set in  $\mathcal{F}$  that has the largest cardinality can be done in  $O(|E|^{d+1}q(d+1))$  time.*

## 2.2 3SUM-Hardness

We show in Theorem 8 that, unless the 3SUM conjecture fails, there exists some polymoid  $P = (E, \mathcal{F})$  so that any algorithm that finds a maximum independent set for  $P$  requires  $\Omega(|E|^{2-\varepsilon})$  time for any constant  $\varepsilon > 0$ .

► **Theorem 8.** *There exists some polynomoid  $P$  whose independence oracle can be decided in time linear in the input size so that finding a maximum independent set of  $P$  is 3SUM-hard.*

**Proof.** Let  $P = (E, \mathcal{F})$  be a polynomoid where  $E$  is a finite set of distinct points in  $\mathbb{R}^2$  and  $\mathcal{F} = \{E_j \subseteq E \mid \text{all points in } E_j \text{ are colinear}\}$ . It is clear that  $P$  is a 2-polynomoid. A maximum independent set of  $P$  corresponds to a line in  $\mathbb{R}^2$  that passes through the most number of points in  $E$ . Hence, it suffices to answer whether there exist three points in  $E$  that are colinear, which is known to be 3SUM-hard [19]. ◀

### 3 Dependent $k$ -Set Packing

In this section, we will present a reduction from the dependent  $k$ -set packing to the maximum independent set.

For each polynomoid  $P = (E, \mathcal{F})$ , we define a *rank* function  $r : 2^E \rightarrow \{0, 1, \dots, |E|\}$  so that  $r(S)$  denotes the cardinality of a largest subset of  $S$  that is contained in  $\mathcal{F}$ . In particular,  $r(E)$  equals the size of a maximum independent set of  $P$ , which can be computed in polynomial time by Theorem 7. More generally, for every  $S \subseteq E$ ,  $r(S)$  equals the size of a maximum independent set of the polynomoid  $Q = (S, 2^S \cap \mathcal{F})$ . Hence, the rank function  $r$  for any subset of  $E$  is computable in polynomial time, assuming that the independence oracle of  $P$  can be decided in polynomial time. Let  $\rho_k(E)$  denote the maximum number of disjoint dependent  $k$ -subsets that  $E$  contains. We claim that  $r(E)$  and  $\rho_k(E)$  can be related as follows, so  $\rho_k(E)$  can be computed no slower than finding  $r(E)$ .

► **Theorem 9.** *Let  $k, d$  be two integers with  $k > d \geq 0$ . For any  $d$ -polynomoid  $P = (E, \mathcal{F})$ ,*

$$\rho_k(E) = \begin{cases} |E| - r(E) & \text{if } r(E) > (1 - 1/k)|E| \\ \lfloor |E|/k \rfloor & \text{otherwise} \end{cases}$$

It suffices to prove Theorem 9 for  $k = d + 1$  because of the observation that a  $d$ -polynomoid is also a  $t$ -polynomoid for every  $t > d$ .

#### 3.1 Case I: $r(E) > (1 - 1/k)|E|$

We prove the first case of Theorem 9 by the following lemma.

► **Lemma 10.** *Let  $d \geq 0$  and  $k = d + 1$  be two integers. For any  $d$ -polynomoid  $P = (E, \mathcal{F})$ , if  $r(E) > (1 - 1/k)|E|$ , then  $\rho_k(E) = |E| - r(E)$ .*

**Proof.** The proof for  $d = 0$  is clear because any 0-polynomoid has  $\mathcal{F} = 2^S$  for some  $S \subseteq E$ . Hence we assume that  $d \geq 1$ , so  $k \geq 2$ . Let  $M$  be a maximum independent set of  $P$ . By definition,  $|M| = r(E)$ . For each element  $x$  in  $E \setminus M$ , we remove  $k - 1$  distinct elements from  $M$  and let  $A_x$  be the set containing  $x$  and the  $k - 1$  removed elements.

▷ **Claim 11.**  $A_x$  is dependent.

**Proof.** Suppose for contradiction that  $A_x \in \mathcal{F}$ , the intersection of  $A_x$  and  $M$  is  $k - 1 = d$ , so  $A_x \cup M \in \mathcal{F}$  by the join property of  $P$ . This violates the maximality of  $M$ . Hence,  $A_x$  is dependent. ◀

Since  $r(E) > (1 - 1/k)|E|$ , we have  $r(E) > (k - 1)(|E| - r(E))$ . So the above grouping procedure can iterate until  $E \setminus M$  is exhausted. Hence, we obtain a collection of  $|E| - r(E)$  dependent  $k$ -sets (not necessarily the largest one), so  $\rho_k(E) \geq |E| - r(E)$ . Note that, if  $\rho_k(E) > |E| - r(E)$ , by the pigeonhole principle, at least one of the  $\rho_k(E)$   $k$ -sets contains elements only from  $M$ . By the hereditary property of  $P$ , such a  $k$ -set must be independent because it is a subset of  $M$ , a contradiction. As a result,  $\rho_k(E) = |E| - r(E)$ . ◀

### 3.2 Case II: $r(E) \leq (1 - 1/k)|E|$

We prove the second case of Theorem 9 by the following lemmas. We will show that, if  $r(E) \leq (1 - 1/k)|E|$ , let  $Z_1, Z_2, \dots, Z_\ell$  be  $k$ -sets of  $E$  and  $W = E \setminus \bigcup_{i \in [\ell]} Z_i$  with  $|W| < k$ , then either  $Z_i$ s are all dependent, or there exist two sets in  $\{Z_i : i \in [\ell]\} \cup W$  whose elements can be exchanged so as to increase the number of dependent sets in  $\{Z_i : i \in [\ell]\}$ .

We begin with a helper lemma.

► **Lemma 12.** *Let  $d \geq 0$  and  $k = d + 1$  be two integers. Let  $P = (E, \mathcal{F})$  be a  $d$ -polynomoid. For any two disjoint  $k$ -subsets  $X, Y \subseteq E$ , if  $r(X \cup Y) \leq 2d$ , then  $X \cup Y$  can be partitioned into two disjoint dependent  $k$ -subsets of  $E$ . This partition can be done in  $O(d^2q(d+1))$  time.*

**Proof.** We begin with the proofs of the following claims.

▷ **Claim 13.** If  $X$  has a subset  $Z \in \mathcal{F}$  with  $|Z| \geq d$ , then  $Z \cup \{y\} \notin \mathcal{F}$  for some  $y \in Y$ .

**Proof.** If such a  $y$  does not exist, then  $Z \cup \{y\} \in \mathcal{F}$  for every  $y \in Y$ . By Lemma 5, there is exactly one maximal independent set that contains  $Z$ . The above two facts imply that  $Z \cup Y \in \mathcal{F}$ . This yields  $r(Z \cup Y) \geq 2d + 1 > r(X \cup Y)$ , a contradiction. ◁

▷ **Claim 14.** If  $X \in \mathcal{F}$ , there exists  $y \in Y$ , for every  $d$ -subset  $Z$  of  $X$ ,  $Z \cup \{y\} \notin \mathcal{F}$ .

**Proof.** By Claim 13,  $X \cup \{y\} \notin \mathcal{F}$  for some  $y \in Y$ . If  $X$  has a  $d$ -subset  $Z$  with  $Z \cup \{y\} \in \mathcal{F}$ , since  $X \cap (Z \cup \{y\}) = Z$ , by the join property of  $P$  we have  $X \cup (Z \cup \{y\}) = X \cup \{y\} \in \mathcal{F}$ , a contradiction. ◁

We are ready to give a proof. If both  $X$  and  $Y$  are dependent, then we are done. Otherwise, we assume w.l.o.g. that  $X \in \mathcal{F}$ . By Claim 14, there exists an  $y^* \in Y$ , for every  $x \in X$ ,  $X \setminus \{x\} \cup \{y^*\} \notin \mathcal{F}$ . If  $Y \setminus \{y^*\} \in \mathcal{F}$ , then by Claim 13 there exists some  $x^* \in X$  so that  $Y \setminus \{y^*\} \cup \{x^*\} \notin \mathcal{F}$ . Otherwise  $Y \setminus \{y^*\} \notin \mathcal{F}$ , then for any  $x \in X$  we have  $Y \setminus \{y^*\} \cup \{x\} \notin \mathcal{F}$ . As a result,  $X \setminus \{x^*\} \cup \{y^*\}$  and  $Y \setminus \{y^*\} \cup \{x^*\}$  both are not in  $\mathcal{F}$  and together partition  $X \cup Y$ . By enumerating all possible  $x^*, y^*$ , it yields the time bound. ◀

We are ready to prove that the swapping procedure can iterate until no sets in  $\{Z_i : i \in [\ell]\}$  are independent.

► **Lemma 15.** *Let  $d \geq 0$ ,  $k = d + 1$ , and  $\ell \geq 1$  be integers. Let  $P = (E, \mathcal{F})$  be a  $d$ -polynomoid. For any subset  $S$  of  $E$ , if  $|S| = \ell k + t$  and  $r(S) \leq \ell d + t$  for some  $t \geq 0$ , then  $S$  contains  $\ell$  disjoint dependent  $k$ -subsets of  $E$ .*

**Proof.** Initially, we partition  $S$  arbitrarily into  $\ell$   $k$ -sets  $Z_1, Z_2, \dots, Z_\ell$  and one  $t$ -set  $W$ . If none of  $Z_i$  for  $i \in [\ell]$  is independent, then we are done. Otherwise,  $Z_i \in \mathcal{F}$  for some  $i \in [\ell]$ . Since  $Z_i \in \mathcal{F}$  and  $|Z_i| = k \geq d$ , by Lemma 5 there is exactly one maximal independent set  $M(Z_i)$  that contains  $Z_i$ . Hence, there are at least  $|S| - |M(Z_i)|$  elements  $x \in S$  so that  $Z_i \cup \{x\} \notin \mathcal{F}$ . We color these elements blue. By the maximality of  $r(S)$ ,  $|M(Z_i)| \leq r(S) \leq \ell d + t$ . Hence,  $|S| - |M(Z_i)| \geq \ell k + t - \ell d - t = \ell$ . There are two cases to discuss.

1.  $W$  contains a blue element  $b$ . By the definition of blue elements,  $Z_i \cup \{b\} \notin \mathcal{F}$ . Let  $y$  be an arbitrary element in  $Z_i$ . Then,  $(Z_i \setminus \{y\}) \cup \{b\} \notin \mathcal{F}$ ; otherwise, by the join property of  $P$

$$\{y\} \cup (Z_i \setminus \{y\}) \in \mathcal{F} \text{ and } (Z_i \setminus \{y\}) \cup \{b\} \in \mathcal{F} \text{ together imply } Z_i \cup \{b\} \in \mathcal{F},$$

a contradiction. If we set  $(Z_i, W) \rightarrow (Z_i \setminus \{y\} \cup \{b\}, W \setminus \{b\} \cup \{y\})$ , then the number of independent sets in  $\{Z_1, Z_2, \dots, Z_\ell\}$  is reduced by one.

2.  $W$  contains no blue elements. Because  $Z_i$  also contains no blue elements, the  $\geq \ell$  blue elements are distributed among  $\{Z_1, Z_2, \dots, Z_\ell\} \setminus \{Z_i\}$ . By the pigeonhole principle, some  $Z_j$  contains at least 2 blue elements. We claim that  $r(Z_i \cup Z_j) \leq 2d$ . Thus, by Lemma 12 we can partition  $Z_i \cup Z_j$  into two dependent  $k$ -sets. Hence, the number of independent sets in  $\{Z_1, Z_2, \dots, Z_\ell\}$  is reduced by at least one.

We prove the claim as follows. Suppose for contradiction that  $r(Z_i \cup Z_j) \geq 2d+1$ . Because  $|Z_i| + |Z_j| = 2(d+1)$ , there exists one element  $x$  so that  $Z_i \cup Z_j \setminus \{x\} \in \mathcal{F}$ . Because there are two blue elements  $b_1, b_2 \in Z_j$  and any independent superset of  $Z_i$  contains no blue elements,  $x \notin Z_j$  and thus  $x \in Z_i$ . By the hereditary property of  $P$ ,  $Z_i \setminus \{x\} \cup \{b_1\} \in \mathcal{F}$ . By the join property of  $P$ ,  $\{x\} \cup (Z_i \setminus \{x\}) \in \mathcal{F}$  and  $(Z_i \setminus \{x\}) \cup \{b_1\} \in \mathcal{F}$  imply that  $Z_i \cup \{b_1\} \in \mathcal{F}$ , a contradiction.

For each case, we can reduce the number of independent sets in  $\{Z_1, Z_2, \dots, Z_\ell\}$  by at least one. Since  $\ell$  is finite, one can always obtain a feasible packing.  $\blacktriangleleft$

Let  $|E| = \ell k + t$  for some  $t \in [0, k)$ . Thus  $\lfloor |E|/k \rfloor = \ell$ . Since  $r(E) \leq (1 - 1/k)(\ell k + t) \leq \ell d + t$ , by Lemma 15 we complete the proof of the second case.

#### 4 Finding a Largest Dependent $k$ -Set Packing in Deterministic Linear Time

In Theorem 8, we have shown that computing  $r(E)$  in general requires  $\Omega(|E|^{2-\varepsilon})$  time for any constant  $\varepsilon > 0$  unless the 3SUM conjecture fails. Hence, to compute the  $k$ -set packing in  $O(E)$  time, one cannot directly compute  $r(E)$  to distinguish which case in Theorem 9 applies.

##### 4.1 The Deterministic Linear-Time Algorithm

In this section, we devise a deterministic linear-time algorithm for finding a largest dependent set packing for any  $d$ -polymoid  $P = (E, \mathcal{F})$ . Let  $d, k$  be two integers with  $k = d + 1$ . Recall that we have to consider only the case of  $k = d + 1$ . Let  $|E| = \ell k + t$  for some  $t \in [0, k)$ . Initially, we partition  $E$  arbitrarily into  $Z_1, Z_2, \dots, Z_\ell$  and  $W$  so that  $|Z_i| = k$  for  $i \in [\ell]$  and  $|W| = t$ . Then we apply the following five steps to find a largest dependent  $k$ -set packing  $\mathcal{D}$ .

1. Set  $\mathcal{A} = \{Z_1, Z_2, \dots, Z_\ell\}$ . Set  $\mathcal{D} = \emptyset$ .
2. If some  $Z_i \in \mathcal{A}$  is dependent, remove  $Z_i$  from  $\mathcal{A}$  and set  $\mathcal{D} = \mathcal{D} \cup \{Z_i\}$ . Otherwise, proceed to the next step.
  - ▶ Remark. This step takes  $O(q(k)|E|/k)$  time.
3. If there exist  $Z_i, Z_j \in \mathcal{A}$  that  $Z_i \cup Z_j$  is dependent, remove  $Z_i$  and  $Z_j$  from  $\mathcal{A}$  and set  $\mathcal{D} = \mathcal{D} \cup \{Z'_i, Z'_j\}$  where  $Z'_i$  and  $Z'_j$  are dependent  $k$ -sets and they partition  $Z_i \cup Z_j$ . Otherwise, proceed to the next step.
  - ▶ Remark. The existence of  $Z'_i$  and  $Z'_j$  is shown in Section 4.2. Because  $|Z_i| \geq d$  and  $|Z_j| \geq d$ ,  $Z_i \cup Z_j$  is independent iff  $Z_i$  and  $Z_j$  belong to the same maximal set. So if  $Z_i \cup Z_j \in \mathcal{F}$  but  $Z_i \cup Z_k \notin \mathcal{F}$ , then  $Z_j \cup Z_k \notin \mathcal{F}$ . Hence, we can keep a list of  $Z$ s so that their pairwise unions are independent sets. For each unpaired  $Y$  outside the pool, pick any  $Z$  in the pool, if  $Z \cup Y \in \mathcal{F}$ , then  $Z' \cup Y \in \mathcal{F}$  for any  $Z'$  in the pool, so expand the pool by adding  $Y$ ; otherwise, pair  $Y, Z$  and throw out  $Y, Z$ . Indeed, this is a generalization of the majority voting [8]. This step takes  $O((q(2k) + k^2q(k))|E|/k)$  time.
4. If  $\mathcal{A} = \emptyset$ , return  $\mathcal{D}$  and stop. Otherwise, find a maximal independent set  $M_{\mathcal{A}}$  of  $P$  that contains all elements in  $\mathcal{A}$  as subsets.

► Remark. If  $\mathcal{A} = \emptyset$ , then  $\mathcal{D}$  is largest possible, so it is a largest dependent  $k$ -set packing. The existence of  $M_{\mathcal{A}}$  is shown in Section 4.2. To find  $M_{\mathcal{A}}$ , let  $Z$  be some element in  $\mathcal{A}$  and find the maximal independent set that contains  $Z$  as a subset in  $O(|E|q(k))$  time by Corollary 6. Since  $|Z| = k \geq d$  and  $Z \in \mathcal{F}$ , we know that  $M_{\mathcal{A}} = M_Z$  by Lemma 5.

5. This final step is reached only if  $|\mathcal{A}| \geq 1$ .

- Case 1:  $r(E) > (1 - 1/k)|E|$ . By Lemma 18,  $M_{\mathcal{A}}$  is a maximum independent set of  $P$ . Since  $M_{\mathcal{A}}$  is given, one can simulate Lemma 10 in  $O(|E|)$  time.
- Case 2:  $r(E) \leq (1 - 1/k)|E|$ . By Lemma 17,  $M_{\mathcal{A}}$  contains all elements in  $\mathcal{A}$ . One can simulate Lemma 15 efficiently as follows.

For each  $Z \in \mathcal{A}$ , to implement Lemma 15, we need to find a set  $Z' \in \{Z_i : i \in [\ell]\} \cup \{W\}$  that contains a sufficient number of elements in  $E \setminus M_{\mathcal{A}}$  (aka “blue elements” in the proof of Lemma 15). Given  $M_{\mathcal{A}}$ , we compute  $E \setminus M_{\mathcal{A}}$  in  $O(|E|)$  time and maintain the locations of these “blue elements” so that for each  $Z \in \mathcal{A}$  we can find the  $Z'$  in  $O(1)$  time.

Hence, the number of swapping steps is  $O(|E|/k)$  and each takes  $O(k^2q(k))$  time (Lemma 12). So this step needs  $O(k|E|q(k))$  time.

As a consequence, the total running time of all steps is bounded by  $O(k|E|q(2k))$ . This completes the proof of Theorem 2 for  $k > d$ .

## 4.2 The Existence Proofs

We will prove the existence proofs required by the algorithm in Section 4.1. In Step 3,  $Z_i, Z_j \in \mathcal{F}$  and  $Z_i \cup Z_j \notin \mathcal{F}$ , so  $r(Z_i \cup Z_j) \leq 2d$  by Lemma 16. Hence, by Lemma 12,  $Z_i \cup Z_j$  can be partitioned into two disjoint dependent sets of size  $k$ . The existence of  $M_{\mathcal{A}}$  required by Step 4 is shown in Lemma 17. Finally, we prove in Lemma 18 that  $M_{\mathcal{A}}$  is a maximum independent set of  $P$  if  $r(E) \geq (1 - 1/k)|E|$  and  $|\mathcal{A}| \geq 1$ .

► **Lemma 16.** *For any  $k$ -subsets  $X, Y$  of  $E$ , if  $X, Y \in \mathcal{F}$  but  $X \cup Y \notin \mathcal{F}$ , then  $r(X \cup Y) \leq 2d$ .*

**Proof.** Let  $S$  be any subset of  $X \cup Y$  with  $|S| = r(X \cup Y)$ . If  $|S| > 2d$ , then there exists  $z \in X \cup Y$  so that  $(X \cup Y) \setminus \{z\} \in \mathcal{F}$ . We assume w.l.o.g. that  $z \in X$  and  $Y \subseteq S$ . Let  $M$  be a maximal independent set of  $P$  that contains  $S$  as a subset. Because  $|M \cap X| \geq |S \cap X| \geq d$ , by the join property of  $P$  we have  $M \cup X \in \mathcal{F}$ . By the maximality of  $M$ ,  $X \subseteq M$ , so  $X \cup Y \subseteq M$ . By the hereditary property of  $P$ ,  $M \in \mathcal{F}$  implies  $X \cup Y \in \mathcal{F}$ . This violates the setting. Therefore,  $r(X \cup Y) \leq 2d$ . ◀

► **Lemma 17.** *In Step 4, such a maximal independent set  $M_{\mathcal{A}}$  of  $P$  always exists.*

**Proof.** The construction of  $\mathcal{A}$  yields that, for any  $X, Y \in \mathcal{A}$ ,  $X \cup Y \in \mathcal{F}$ . By Lemma 5, there is a unique maximal independent set  $M_X$  (resp.  $M_{X \cup Y}$ ) that contains  $X$  (resp.  $X \cup Y$ ) as a subset. The uniqueness of  $M_X$  and  $M_{X \cup Y}$  implies that  $M_X = M_{X \cup Y}$ . Similarly,  $M_Y = M_{X \cup Y}$ . Hence,  $M_X = M_Y$ . Because this fact applies to every pair of elements in  $\mathcal{A}$ , there is a maximal independent set  $M_{\mathcal{A}}$  that contains every element in  $\mathcal{A}$  as a subset. ◀

► **Lemma 18.** *In Step 4, if  $r(E) > (1 - 1/k)|E|$  and  $|\mathcal{A}| \geq 1$ , then  $M_{\mathcal{A}}$  is a maximum independent set of  $P$ .*

**Proof.** We begin with the proofs that, for any maximal independent set  $M$  of  $P$ , each removed  $Z_i$  in Steps 2 and 3 contains at least one element outside  $M$  on average.

## 84:12 Dependent $k$ -Set Packing on Polymoids

▷ **Claim 19.** Let  $M$  be any maximal independent set of  $P$ . In Step 2, for each removed  $Z_i$ ,  $Z_i$  contains at least one element in  $E \setminus M$ .

*Proof.* Because  $Z_i \notin \mathcal{F}$  and  $M \in \mathcal{F}$ , by the hereditary property of  $P$ ,  $Z_i$  is not a subset of  $M$ . Hence,  $Z_i$  contains at least one element outside  $E \setminus M$ . ◁

▷ **Claim 20.** Let  $M$  be any maximal independent set of  $P$ . In Step 3, for each pair of removed  $Z_i$  and  $Z_j$ ,  $Z_i \cup Z_j$  contains at least two elements in  $E \setminus M$ .

*Proof.* By Lemma 16, we know that  $r(Z_i \cup Z_j) \leq 2d$ . If  $M \cap (Z_i \cup Z_j) > 2d$ , then  $r(Z_i \cup Z_j) > 2d$ , a contradiction. Hence,  $M \cap (Z_i \cup Z_j) \leq 2d$ , as desired. ◁

We are ready to give a proof. Let  $S$  be a subset of  $E$  with  $|S| = r(E)$ , and let

$$U_{\mathcal{A}} = W \cup \bigcup_{Z \in \mathcal{A}} Z \text{ and } U_B = E \setminus U_{\mathcal{A}}.$$

By Claims 19 and 20, we have

$$\frac{|S \cap U_B|}{|U_B|} \leq \frac{k-1}{k}. \quad (1)$$

By restating  $r(E) > (1 - 1/k)|E|$ , we get

$$\frac{|S \cap E|}{|E|} > \frac{k-1}{k}. \quad (2)$$

Combining (1), (2), and the average argument, it yields that

$$\frac{|S \cap U_{\mathcal{A}}|}{|U_{\mathcal{A}}|} > \frac{k-1}{k}. \quad (3)$$

To satisfy the inequality (3), if  $|\mathcal{A}| \geq 1$ , either  $S$  contains  $U_{\mathcal{A}} \setminus W$  as a subset or  $S$  does not contain  $U_{\mathcal{A}} \setminus W$  as a subset. The former implies that  $S = M_{\mathcal{A}}$  due to the uniqueness of  $M_{\mathcal{A}}$  (Lemma 5), as desired. Note that  $|S \cap (U_{\mathcal{A}} \setminus W)| \leq k-2$ ; otherwise,  $S \cup M_{\mathcal{A}} \in \mathcal{F}$  due to the join property of  $P$  and thus  $S = M_{\mathcal{A}}$  by the maximality of  $S$  and  $M_{\mathcal{A}}$ . The latter cannot hold because  $|S \cap (U_{\mathcal{A}} \setminus W)| \leq k-2$  and  $0 \leq |W| \leq k-1$  implies that

$$\frac{k-1}{k} < \frac{|S \cap U_{\mathcal{A}}|}{|U_{\mathcal{A}}|} = \frac{|S \cap W| + |S \cap (U_{\mathcal{A}} \setminus W)|}{|W| + |U_{\mathcal{A}} \setminus W|} \leq \frac{|W| + k-2}{|W| + k|\mathcal{A}|} \leq \frac{2k-3}{2k-1},$$

which cannot hold for positive  $k$ . ◀

## 5 Conclusion

We obtain a complexity trichotomy result for the dependent  $k$ -set packing problem on  $d$ -polymoids. For each of the three categories, our algorithm is optimal. It may worth noting that the running time of the algorithm for the case of  $k > d$  can be reduced by a factor of  $k$  by group testing [30, 22], which will be introduced in the full version of this manuscript. Though this yields a constant-factor improvement, it may affect the performance of real applications.

## References

- 1 Ron Aharoni. Matchings in  $n$ -partite  $n$ -graphs. *Graphs Comb.*, 1(1):303–304, 1985.
- 2 Ron Aharoni, Eli Berger, Dani Kotlar, and Ran Ziv. Degree conditions for matchability in 3-partite hypergraphs. *J. Graph Theory*, 87(1):61–71, 2018.
- 3 Ron Aharoni and Penny Haxell. Hall’s theorem for hypergraphs. *J. Graph Theory*, 35(2):83–88, 2000.
- 4 N. Alon. A note on the decomposition of graphs into isomorphic matchings. *Acta Mathematica Hungarica*, 42(3-4):221–223, September 1983.
- 5 János Barát and Dániel Gerbner. Edge-decomposition of graphs into copies of a tree with four edges. *Electron. J. Comb.*, 21(1):1, 2014.
- 6 A. Bialostocki and Y. Roditty. 3K2-decomposition of a graph. *Acta Mathematica Hungarica*, 40(3-4):201–208, September 1982.
- 7 Ahmad Biniiaz, Anil Maheshwari, Subhas C. Nandy, and Michiel H. M. Smid. An optimal algorithm for plane matchings in multipartite geometric graphs. In *Algorithms and Data Structures – 14th International Symposium (WADS)*, pages 66–78. Springer, 2015.
- 8 Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118. Kluwer Acad. Publishers, 1991.
- 9 Andries Brouwer and R. Wilson. The decomposition of graphs into ladder graphs, January 1980.
- 10 Krzysztof Brys and Zbigniew Lonc. Polynomial cases of graph decomposition: A complete solution of Holyer’s problem. *Discret. Math.*, 309(6):1294–1326, 2009.
- 11 Rainer Burkard, Mauro Dell’Amico, and Silvano Martello. *Assignment problems: revised reprint*. SIAM, 2012.
- 12 Ho Yee Cheung, Lap Chi Lau, and Kai Man Leung. Graph connectivities, network coding, and expander graphs. *SIAM J. Comput.*, 42(3):733–751, 2013.
- 13 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
- 14 David E Daykin and Roland Häggkvist. Degrees giving independent edges in a hypergraph. *Bulletin of the Australian Mathematical Society*, 23(1):103–109, 1981.
- 15 Dorit Dor and Michael Tarsi. Graph decomposition is NP-complete: A complete proof of Holyer’s conjecture. *SIAM J. Comput.*, 26(4):1166–1187, 1997.
- 16 M.E. Dyer and A.M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.
- 17 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B:67–72, 1965.
- 18 O. Favaron, Z. Lonc, and M. Truszczynski. Decomposition of graphs into graphs with three edges. *Ars Combin.*, 20:125–146, 1985.
- 19 Anka Gajentaan and Mark H. Overmars. On a class of  $O(n^2)$  problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.
- 20 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 21 Eunice Gogo Mphako. Circuit and cocircuit partitions of binary matroids. *Czechoslovak Mathematical Journal*, 56(1):19–25, 2006.
- 22 Oded Goldreich. Combinatorial property testing (a survey). *Randomization Methods in Algorithm Design*, 43:45–59, 1999.
- 23 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
- 24 Philip Hall. On representatives of subsets. *Classic Papers in Comb.*, pages 58–62, 1987.
- 25 Jie Han and Peter Keevash. Finding perfect matchings in dense hypergraphs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2366–2377. SIAM, 2020.
- 26 Jie Han and Andrew Treglown. The complexity of perfect matchings and packings in dense hypergraphs. *J. Comb. Theory, Ser. B*, 141:72–104, 2020.

- 27 Penny E. Haxell. A condition for matchability in hypergraphs. *GCOM.*, 11(3):245–248, 1995.
- 28 Ian James Holyer. The computational complexity of graph theory problems, 1981.
- 29 Cor A. J. Hurkens and Alexander Schrijver. On the size of systems of sets every  $t$  of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discret. Math.*, 2(1):68–72, 1989.
- 30 Frank K Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, 1972.
- 31 Per M. Jensen and Bernhard Korte. Complexity of matroid property algorithms. *SIAM J. Comput.*, 11(1):184–190, 1982.
- 32 Viggo Kann. Maximum bounded 3-dimensional matching is MaxSNP-complete. *Inf. Process. Lett.*, 37(1):27–35, 1991.
- 33 Peter Keevash, Fiachra Knox, and Richard Mycroft. Polynomial-time perfect matchings in dense hypergraphs. In *Symp. on Theory of Computing (STOC)*, pages 311–320. ACM, 2013.
- 34 Imdadullah Khan. Perfect matchings in 3-uniform hypergraphs with large vertex degree. *SIAM J. Discret. Math.*, 27(2):1021–1039, 2013.
- 35 Imdadullah Khan. Perfect matchings in 4-uniform hypergraphs. *J. Comb. Theory, Ser. B*, 116:333–366, 2016.
- 36 Silvio Micali and Vijay V. Vazirani. An  $O(\sqrt{VE})$  algorithm for finding maximum matching in general graphs. In *Symp. on Fnd. of Computer Science (FOCS)*, pages 17–27. IEEE, 1980.
- 37 James G. Oxley. *Matroid theory*. Oxford University Press, 1992.
- 38 Vojtech Rödl, Andrzej Rucinski, and Endre Szemerédi. Perfect matchings in uniform hypergraphs with large minimum degree. *Eur. J. Comb.*, 27(8):1333–1349, 2006.
- 39 Paul D. Seymour. Packing and covering with matroid circuits. *J. Comb. Theory, Ser. B*, 28(2):237–242, 1980.
- 40 Nicolas Teypez and Christophe Rapine. Graph decomposition into paths under length constraints. Technical report, UniK University Graduate Center, University of Oslo, 2004.
- 41 D.J.A. Welsh. Euler and bipartite matroids. *Journal of Comb. Theory*, 6(4):375–377, 1969.
- 42 Douglas B. West. *Combinatorial Mathematics*. CUP, 2020.
- 43 Sanming Zhou. Minimum partition of an independence system into independent sets. *Discret. Optim.*, 6(1):125–133, 2009.

## **A** Hardness Reduction

In this section, we prove that the dependent  $k$ -set packing for  $d$ -polynomoids with  $d \geq k$  is as hard as hypergraph matchings in general. We reduce the matching problem for  $k$ -uniform hypergraphs to the dependent  $k$ -set packing problem on  $d$ -polynomoids with  $d \geq k$  in Theorem 21. The other direction is clear because the latter problem is a special case of the former one. This completes the proof of Theorem 2 for  $k \leq d$ .

► **Theorem 21.** *For any integers  $d \geq k \geq 1$ , there exists a  $d$ -polynomoid  $P = (E, \mathcal{F})$  so that dependent  $k$ -set packing on  $P$  is as hard as matchings on  $k$ -uniform hypergraphs.*

**Proof.** We prove this lemma by showing a reduction from the perfect matching for  $k$ -uniform graphs to the dependent  $k$ -set packing for  $P$ . Let  $G = (V, \mathcal{E})$  be a  $k$ -uniform hypergraph. Let  $P = (V, \mathcal{F})$  be a  $d$ -polynomoid where  $\mathcal{F} = \{\text{all subsets of } V \text{ with size at most } k\} \setminus \mathcal{E}$ . We now show that  $P$  is a  $d$ -polynomoid. Note that each set in  $\mathcal{F}$  has size at most  $k$ . By the definition of  $\mathcal{F}$ , all subsets of  $V$  with size at most  $k - 1$  are in  $\mathcal{F}$  so  $P$  satisfies the hereditary property. When  $k = d$ , two sets  $A, B$  in  $\mathcal{F}$  have  $|A \cap B| \geq d$  if and only if  $A = B$ . When  $k < d$ , two sets  $A, B$  in  $\mathcal{F}$  have  $|A \cap B| \geq d$  cannot happen. Hence,  $P$  satisfies the join property. Observe that a hyperedge is in  $G$  if and only if it is a dependent set of  $P$ . Hence, finding a dependent  $k$ -set packing for  $P$  is equivalent to finding a matching for  $G$ . ◀

## B More Applications

In this section, we present more applications of our results that are not covered in Section 1.1.

1. Let  $G = (V, E)$  be an  $m$ -edge undirected simple graph. Let  $P = (E, \mathcal{F})$  so that  $\mathcal{F} = \{A \subseteq E : \text{there exists a triangle in } G \text{ that contains all edges in } A\}$ . One may verify that  $P$  is a 2-polymoid. By Theorem 2, we obtain a deterministic linear-time algorithm for the dependent  $k$ -set packing for  $P$ , which is equivalent to partitioning the edge set of  $G$  into subsets, each of which edge-induces a 3-edge forest. In addition, we obtain the following sufficient and necessary condition:

► **Corollary 22.** *An  $m$ -edge undirected simple graph  $G$  can be edge-partitioned into 3-edge forests if and only if  $m$  is a multiple of 3 and  $G$  is not a triangle.*

In [5], they give a sufficient and necessary condition to partition the edge set of the given graph into a designated four-edge tree for highly-edge-connected graphs.

2. Let  $G = (V, E)$  be a complete multipartite graph. Let  $P = (V, \mathcal{F})$  so that  $\mathcal{F} = \{A \subseteq V : \text{all vertices in } A \text{ are from the same partite set}\}$ . One may verify that  $P$  is a 1-polymoid. By Theorem 2, we obtain the following sufficient and necessary condition:

► **Corollary 23.** *An  $n$ -vertex undirected simple complete multipartite graph  $G$  has a perfect matching if and only if  $n$  is a multiple of 2 and the number of vertices in a largest partite set is at most  $n/2$ .*

By an argument in [7], Corollary 23 suffices to prove:

► **Corollary 24.** *Given a set of points in  $\mathcal{R}^2$  in general position where each point has a color in  $[c]$ , one can group the points into pairs so that the line segment joining the points in a pair does not cross that of another pair and the points in each pair have different colors if and only if  $n$  is a multiple of 2 and for each color  $i \in [c]$  the number of points of color  $i$  is at most  $n/2$ .*

## C Omitted Proofs

► **Lemma 25.** *Let  $G = (V, E)$  be an undirected simple graph consisting of six edges. If  $E$  can be partitioned into  $E_1, E_2$  so that  $E_1$  edge-induces a triangle and  $E_2$  edge-induces a non-star graph, then  $E$  can also be partitioned into two subsets so that each subset edge-induces a 3-edge linear forest.*

**Proof.** By definition,  $E_2$  edge-induces either a triangle or a 3-edge linear forest. Suppose that  $E_2$  edge-induces a triangle. For any  $E' \subseteq E$ , let  $V(E')$  denote the set of the end-vertices of edges in  $E'$ . Since  $G$  is simple,  $V(E_1)$  and  $V(E_2)$  have at most one vertex in common. Hence, there exist  $e_1 \in E_1, e_2 \in E_2$  so that  $V(\{e_1\}) \cap V(E_2) = \emptyset$  and  $V(\{e_2\}) \cap V(E_1) = \emptyset$ . This yields that  $E_1 \cup \{e_2\} \setminus \{e_1\}$  and  $E_2 \cup \{e_1\} \setminus \{e_2\}$  both edge-induce  $P_3 \cup P_2$ , i.e., two vertex-disjoint paths of 3 vertices and 2 vertices.

Suppose that  $E_2$  edge-induces a 3-edge linear forest. Let  $e_2$  be an edge in  $E_2$  so that  $E_2 \setminus \{e_2\} = 2P_2$ , i.e., two edges with no end-vertices in common. Since  $G$  is simple and  $E_1$  edge-induces a complete graph,  $|V(\{e_2\}) \cap V(E_1)| \leq 1$ . Hence,  $E_1 \cup \{e_2\}$  has at most one vertex of degree  $\geq 3$  and  $e_2$  is not an edge in any cycle of  $E_1 \cup \{e_2\}$ . Let  $e_1$  be any edge in  $E_1$  incident with the maximum degree vertex in  $E_1 \cup \{e_2\}$ . Thus,  $E_1 \cup \{e_2\} \setminus \{e_1\}$  has max degree  $\leq 2$  and contains no cycle, i.e., a linear forest. On the other hand, because the union of  $2P_2$  and any other edge also has max degree  $\leq 2$  and contains no cycle,  $E_2 \cup \{e_1\} \setminus \{e_2\}$  also is a linear forest. ◀



# Exponential Lower Bounds for Threshold Circuits of Sub-Linear Depth and Energy

Kei Uchizawa ✉

Graduate School of Science and Engineering, Yamagata University, Yonezawa-shi Yamagata, Japan

Haruki Abe

Graduate School of Science and Engineering, Yamagata University, Yonezawa-shi Yamagata, Japan

---

## Abstract

---

In this paper, we investigate computational power of threshold circuits and other theoretical models of neural networks in terms of the following four complexity measures: size (the number of gates), depth, weight and energy. Here, the energy of a circuit measures sparsity of their computation, and is defined as the maximum number of gates outputting non-zero values taken over all the input assignments.

As our main result, we prove that any threshold circuit  $C$  of size  $s$ , depth  $d$ , energy  $e$  and weight  $w$  satisfies  $\log(\text{rk}(M_C)) \leq ed(\log s + \log w + \log n)$ , where  $\text{rk}(M_C)$  is the rank of the communication matrix  $M_C$  of a  $2n$ -variable Boolean function that  $C$  computes. Thus, such a threshold circuit  $C$  is able to compute only a Boolean function of which communication matrix has rank bounded by a product of logarithmic factors of  $s, w$  and linear factors of  $d, e$ . This implies an exponential lower bound on the size of even sublinear-depth and sublinear-energy threshold circuit. For example, we can obtain an exponential lower bound  $s = 2^{\Omega(n^{1/3})}$  for threshold circuits of depth  $n^{1/3}$ , energy  $n^{1/3}$  and weight  $2^{o(n^{1/3})}$ . We also show that the inequality is tight up to a constant factor when the depth  $d$  and energy  $e$  satisfies  $ed = o(n/\log n)$ .

For other models of neural networks such as a discretized ReLU circuits and discretized sigmoid circuits, we define energy as the maximum number of gates outputting non-zero values. We then prove that a similar inequality also holds for a discretized circuit  $C$ :  $\text{rk}(M_C) = O(ed(\log s + \log w + \log n)^3)$ . Thus, if we consider the number gates outputting non-zero values as a measure for sparse activity of a neural network, our results suggest that larger depth linearly helps neural networks to acquire sparse activity.

**2012 ACM Subject Classification** Theory of computation → Models of computation

**Keywords and phrases** Circuit complexity, disjointness function, equality function, neural networks, threshold circuits, ReLU circuits, sigmoid circuits, sparse activity

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.85

**Related Version** *Full Version*: <https://arxiv.org/abs/2107.00223>

**Funding** *Kei Uchizawa*: This work was supported by JSPS KAKENHI Grant Number JP22K11897.

**Acknowledgements** We thank the anonymous reviewers for their careful reading and helpful comments.

## 1 Introduction

**Background.** DiCarlo and Cox argued that constructing good internal representations is crucial to perform visual information processing, such as object recognition, for neural networks in the brain [5]. Here, an internal representation is described by a vector in a very high dimensional space, where each axis is one neuron's activity and the dimensionality equals to the number (e.g.,  $\sim 1$  million) of neurons in a feedforward neural network. They call a representation good if, for a given pair of two images that are hard to distinguish at the input space, there exist representations for them that are easy to separate by simple



© Kei Uchizawa and Haruki Abe;  
licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 85; pp. 85:1–85:15

Leibniz International Proceedings in Informatics



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

classifiers such as a linear classifier. While such internal representations are likely to play fundamental role in information processing in the brain, it is also known that a neuron needs relatively high energy to be active [18, 27], and hence neural networks are forced to acquire representations supported by only a small number of active neurons [7]. These observations pose a question: for what information processing can neural networks construct good internal representations?

In the paper [42], Uchizawa *et al.* address the question from the viewpoint of circuit complexity. More formally, they employed threshold circuits as a model of neural networks [23, 24, 28, 31, 34, 35, 36], and introduced a complexity measure, called energy complexity, for sparsity of their internal representations. A threshold circuit is a feedforward logic circuit whose basic computational element computes a linear threshold function, and energy of a circuit is defined as the maximum number of internal gates outputting ones over all the input assignments. (See also [6, 16, 33, 37, 45] for studies on energy complexity of other types of logic circuits). Uchizawa *et al.* then show that the energy complexity is closely related to the rank of linear decision trees. In particular, they prove that any linear decision tree of  $l$  leaves can be simulated by a threshold circuit of size  $O(l)$  and energy  $O(\log l)$ . Thus, even logarithmic-energy threshold circuits have certain computational power: any linear decision tree of polynomial number of leaves can be simulated by a polynomial-size and logarithmic-energy threshold circuit.

Following the paper [42], a sequence of papers show relations among other major complexity measures such as size (the number of gates), depth, weight and fan-in [22, 38, 39, 43, 41, 40, 44]. In particular, Uchizawa and Takimoto [43] showed that any threshold circuit  $C$  of depth  $d$  and energy  $e$  requires size  $s = 2^{\Omega(n/e^d)}$  if  $C$  computes a high bounded-error communication complexity function such as Inner-Product function. Even for low communication complexity functions, an exponential lower bound on the size is known for constant-depth threshold circuits: any threshold circuit  $C$  of depth  $d$  and energy  $e$  requires size  $s = 2^{\Omega(n/e2^{e+d} \log^e n)}$  if  $C$  computes the parity function [41]. These results provide exponential lower bounds if the depth is constant and energy is sub-linear [43] or sub-logarithmic [41], while both Inner-Product function and Parity function are computable by linear-size, constant-depth, and linear-energy threshold circuits. Thus these results imply that the energy complexity strongly related to representational power of threshold circuits. However these lower bounds break down when we consider threshold circuits of larger depth and energy, say, non-constant depth and sub-linear energy.

**Our Results for Threshold Circuits.** In this paper, we prove that simple Boolean functions are hard even for sub-linear depth and sub-linear energy threshold circuits. Let  $C$  be a threshold circuit with Boolean input variables  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  and  $\mathbf{y} = (y_1, y_2, \dots, y_n)$ . A communication matrix  $M_C$  of  $C$  is a  $2^n \times 2^n$  matrix where each row (resp., each column) is indexed by an assignment  $\mathbf{a} \in \{0, 1\}^n$  to  $\mathbf{x}$  (resp.,  $\mathbf{b} \in \{0, 1\}^n$  to  $\mathbf{y}$ ), and the value  $M_C[\mathbf{a}, \mathbf{b}]$  is defined to be the output of  $C$  given  $\mathbf{a}$  and  $\mathbf{b}$ . We denote by  $rk(M_C)$  the rank of  $M_C$  over  $\mathbb{F}_2$ . Our main result is the following relation among size, depth energy and weight.

► **Theorem 1.** *Let  $s, d, e$  and  $w$  be integers satisfying  $2 \leq s, d, 10 \leq e, 1 \leq w$ . If a threshold circuit  $C$  computes a Boolean function of  $2n$  variables, and has size  $s$ , depth  $d$ , energy  $e$  and weight  $w$ , then it holds that*

$$\log(rk(M_C)) \leq ed(\log s + \log w + \log n). \quad (1)$$

The theorem implies exponential lower bounds for sub-linear depth and sub-linear energy threshold circuits. As an example, let us consider a Boolean function  $CD_n$  defined as follows: For a  $2n$  input variables  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$ ,

$$CD_n(\mathbf{x}, \mathbf{y}) = \bigvee_{i=1}^n x_i \wedge y_i.$$

We note that  $CD_n$  is a biologically motivated Boolean function: Maass [21] defined  $CD_n$  to model coincidence detection or a pattern matching, and Lynch and Musco [19] introduced a related problem, called Filter problem, for studying theoretical aspect of spiking neural networks. Since  $CD_n$  is the complement of the disjointness function and has rank  $2^n$ , the theorem implies that

$$n \leq ed(\log s + \log w + \log n) \quad (2)$$

holds if a threshold circuit  $C$  computes  $CD_n$ . Arranging Eq. (2), we can obtain a lower bound  $2^{n/(ed)}/(wn) \leq s$  which is exponential in  $n$  if both  $d$  and  $e$  are sub-linear and  $w$  is sub-exponential. For example, we can obtain an exponential lower bound  $s = 2^{\Omega(n^{1/3})}$  even for threshold circuits of depth  $n^{1/3}$ , energy  $n^{1/3}$  and weight  $2^{o(n^{1/3})}$ . We can obtain similar lower bounds for the Inner-Product function and the equality function, since they have linear rank.

Comparing the lower bound  $s = 2^{\Omega(n/e^d)}$  given in [43] to ours, our lower bound is meaningful only for sub-exponential weight, but improves on it in two-fold: the lower bound is exponential even if  $d$  is sub-linear, and provide a nontrivial lower bound for Boolean functions with much weaker condition: Threshold circuits need exponential size even for Boolean functions of the standard rank  $\Omega(n)$ .

Threshold circuits have received considerable attention in circuit complexity, and a number of lower bound arguments have developed for threshold circuits under some restrictions on computational resources including size, depth, energy and weight [1, 2, 3, 9, 10, 13, 15, 22, 26, 30, 32, 41, 43, 44]. However, the arguments for lower bounds are designated for constant-depth threshold circuits, and hence cannot provide meaningful ones when the depth is not constant. In particular,  $CD_n$  is computable by a depth-2 and linear-size threshold circuit. Thus, directly applying known techniques are unlikely to yield an exponential lower bound for  $CD_n$ .

To complement Theorem 1, we also show that the lower bound is tight up to a constant factor if the product of  $e$  and  $d$  are small:

► **Theorem 2.** *For any integers  $e$  and  $d$  such that  $2 \leq e$  and  $2 \leq d$ ,  $CD_n$  is computable by a threshold circuit of size*

$$s \leq (e-1)(d-1) \cdot 2^{\frac{n}{(e-1)(d-1)}}.$$

*depth  $d$ , energy  $e$  and weight*

$$w \leq \left( \frac{n}{(e-1)(d-1)} \right)^2.$$

Substituting  $s, d, e$  and  $w$  of a threshold circuit given in Theorem 2 to the right hand side of Eq. (2), we have

$$\begin{aligned} & ed(\log s + \log w + \log n) \\ & \leq ed \left( \frac{n}{(e-1)(d-1)} + \log(e-1)(d-1) + \log \left( \frac{n}{(e-1)(d-1)} \right)^2 + \log n \right) \\ & \leq 4n + O(ed \log n), \end{aligned}$$

which almost matches the left hand side of Eq. (2) if  $ed = o(n/\log n)$ . Thus, Theorem 1 neatly captures the computational aspect of threshold circuits computing  $CD_n$ . Recall that any linear decision tree of polynomial number of leaves can be simulated by a polynomial-size and logarithmic-energy threshold circuit [42]. Also, it is known that any Boolean function is computable by a threshold circuit of depth two and energy one if an exponential size is allowed [22]. Thus, we believe that the situation  $ed = o(n/\log n)$  is not too restrictive. We also show that the lower bound is also tight for the equality function.

**Our Result for Discretized Circuits.** Besides threshold circuits, we consider other well-studied models of neural network, where an activation function and weights of a computational element are discretized (such as, discretized sigmoid or ReLU circuits). The size, depth, energy and weight are important parameters also for artificial neural networks. The size and depth are major topics on success of deep learning. The energy is related to important techniques for deep learning method such as regularization, sparse coding, or sparse autoencoder [11, 17, 25]. The weight resolution is closely related to chip resources in neuromorphic hardware systems [29], and quantization schemes received attention [4, 12].

We define energy for a discretized circuit as the maximum number of gates outputting non-zero values, and show that any discretized circuit can be simulated by a threshold circuit with a moderate increase in size, depth, energy, and weight. Consequently, combining with Theorem 1, we can show that its rank is bounded by a product of the polylogarithmic factors of  $s, w$  and linear factors of  $d, e$  for discretized circuits. For example, we can obtain the following proposition for discretized sigmoid circuits:

► **Theorem 3.** *If a discretized sigmoid circuit  $C$  of size  $s$ , depth  $d$ , energy  $e$ , and weight  $w$  computes a Boolean function  $f$ , then it holds that*

$$\log(\text{rk}(M_C)) = O(ed(\log s + \log w + \log n)^3).$$

Maass, Schnitger and Sontag [20] showed that a sigmoid circuit could be simulated by a threshold circuit, but their simulation was optimized to be depth-efficient and did not consider energy. Thus, their result does not fit into our purpose.

Theorems 1 and 3 imply that a threshold circuit or discretized circuit are able to compute a Boolean function of bounded rank. Thus, we can consider these theorems as bounds on corresponding concept classes. According to the bound,  $c$  times larger depth is comparable to  $2^c$  times larger size. Thus, large depth could enormously help neural networks to increase its expressive power. Also, the bound suggests that increasing depth could also help a neural network to acquire sparse activity when we have hardware constraints on both the number of neurons and the weight resolution. These observations may shed some light on the reason for the success of deep learning.

**Organization.** The rest of the paper is organized as follows. In Section 2, we define terms needed for analysis. In Section 3, we present our main lower bound result. In Section 4, we show the tightness of the lower bound. In Section 5, we show a bound for discretized circuits. In Section 6, we conclude with some remarks.

## 2 Preliminaries

For an integer  $n$ , we denote by  $[n]$  a set  $\{1, 2, \dots, n\}$ . The base of the logarithm is two unless stated otherwise. In Section 2.1, we define terms on threshold circuits and discretized circuits. In Section 2.2, we define communication matrix, and give some known facts.

## 2.1 Circuit Model

In Sections 2.1.1 and 2.1.2, we give definitions of threshold and discretized circuits, respectively.

### 2.1.1 Threshold Circuits

Let  $k$  be a positive integer. A *threshold gate*  $g$  with  $k$  input variables  $\xi_1, \xi_2, \dots, \xi_k$  has weights  $w_1, w_2, \dots, w_k$ , and a threshold  $t$ . We define the output  $g(\xi_1, \xi_2, \dots, \xi_k)$  of  $g$  as

$$g(\xi_1, \xi_2, \dots, \xi_k) = \text{sign} \left( \sum_{i=1}^k w_i \xi_i - t \right) = \begin{cases} 1 & \text{if } t \leq \sum_{i=1}^k w_i \xi_i; \\ 0 & \text{otherwise} \end{cases}$$

To evaluate the weight resolution, we assume single synaptic weight to be discrete, and that  $w_1, w_2, \dots, w_n$  are integers. The *weight*  $w_g$  of  $g$  is defined as the maximum of the absolute values of  $w_1, w_2, \dots, w_k$ . In other words, we assume that  $w_1, w_2, \dots, w_k$  are  $O(\log w_g)$ -bit coded discrete values. Throughout the paper, we allow a gate to have both positive and negative weights, although biological neurons are either excitatory (all the weights are positive) or inhibitory (all the weights are negative). As mentioned in [21], this relaxation has basically no impact on circuit complexity investigations, unless one cares about constant blowup in computational resources.

A *threshold circuit*  $C$  is a combinatorial circuit consisting of threshold gates, and is expressed by a directed acyclic graph. The nodes of in-degree 0 correspond to input variables, and the other nodes correspond to gates. Let  $G$  be a set of the gates in  $C$ . For each gate  $g \in G$ , the *level* of  $g$ , denoted by  $\text{lev}(g)$ , is defined as the length of a longest path from an input variable to  $g$  on the underlying graph of  $C$ . For each  $l \in [d]$ , we define  $G_l$  as a set of gates in the  $l$ th level:  $G_l = \{g \in G \mid \text{lev}(g) = l\}$ .

In this paper, we consider a threshold circuit  $C$  for a Boolean function  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ . Thus,  $C$  has  $2n$  Boolean input variables  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  and  $\mathbf{y} = (y_1, y_2, \dots, y_n)$ , and a unique output gate, denoted by  $g^{\text{clf}}$ , which is a linear classifier separating internal representations given by the gates in the lower levels (possibly together with input variables). Consider a gate  $g$  in  $C$ . Let  $w_1^x, w_2^x, \dots, w_n^x$  (resp.,  $w_1^y, w_2^y, \dots, w_n^y$ ) be the weights for  $x_1, x_2, \dots, x_n$  (resp.,  $y_1, y_2, \dots, y_n$ ), and  $t_g$  be threshold of  $g$ . For each gate  $h$  directed to  $g$ , let  $w_{h,g}$  be a weight of  $g$  for the output of  $h$ . Then the output  $g(\mathbf{x}, \mathbf{y})$  of  $g$  is defined as

$$g(\mathbf{x}, \mathbf{y}) = \text{sign}(p_g(\mathbf{x}, \mathbf{y}) - t_g)$$

where  $p_g(\mathbf{x}, \mathbf{y})$  denotes a potentials of  $g$  invoked by the input variables and gates:

$$p(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n w_i^x x_i + \sum_{i=1}^n w_i^y y_i + \sum_{l=1}^{\text{lev}(g)-1} \sum_{h \in G_l} w_{h,g} h(\mathbf{x}, \mathbf{y}).$$

We sometimes write  $p_g^x(\mathbf{x})$  (resp.,  $p_g^y(\mathbf{y})$ ) for the potential invoked by  $\mathbf{x}$  (resp.,  $\mathbf{y}$ ):

$$p_g^x(\mathbf{x}) = \sum_{i=1}^n w_i^x x_i \quad \text{and} \quad p_g^y(\mathbf{y}) = \sum_{i=1}^n w_i^y y_i.$$

Although the inputs to  $g$  are not only  $\mathbf{x}$  and  $\mathbf{y}$  but the outputs of gates in the lower levels, we write  $g(\mathbf{x}, \mathbf{y})$  for the output of  $g$ , because  $\mathbf{x}$  and  $\mathbf{y}$  inductively decide the output of  $g$ . We say that  $C$  *computes* a Boolean function  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  if  $g^{\text{clf}}(\mathbf{a}, \mathbf{b}) = f(\mathbf{a}, \mathbf{b})$  for every  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ .

Let  $C$  be a threshold circuit. We define *size*  $s$  of  $C$  as the number of the gates in  $C$ , and *depth*  $d$  of  $C$  as the level of  $g^{\text{clf}}$ . We define the *energy*  $e$  of  $C$  as

$$e = \max_{(\mathbf{a}, \mathbf{b}) \in \{0,1\}^{2n}} \sum_{g \in G} g(\mathbf{a}, \mathbf{b}).$$

We define *weight*  $w$  of  $C$  as the maximum of the weights of the gates in  $C$ :  $w = \max_{g \in G} w_g$ .

### 2.1.2 Discretized Circuits

Let  $\varphi$  be an activation function. Let  $\delta$  be a discretizer that maps a real number to a number representable by a bitwidth  $b$ . We define a discretized activation function  $\delta \circ \varphi$  as a composition of  $\varphi$  and  $\delta$ , that is,  $\delta \circ \varphi(x) = \delta(\varphi(x))$  for any number  $x$ . We say that  $\delta \circ \varphi$  has *silent range for an interval*  $I$  if  $\delta \circ \varphi(x) = 0$  if  $x \in I$ , and  $\delta \circ \varphi(x) \neq 0$ , otherwise. For example, if we use the ReLU function as the activation function  $\varphi$ , then  $\delta \circ \varphi$  has silent range for  $I = (-\infty, 0]$  for any discretizer  $\delta$ . If we use the sigmoid function as the activation function  $\varphi$  and linear partition as discretizer  $\delta$ , then  $\delta \circ \varphi$  has silent range for  $I = (-\infty, t_{\max}]$  where  $t_{\max} = \ln(1/(2^b - 1))$  where  $\ln$  is the natural logarithm.

Let  $\delta \circ \varphi$  be a discretized activation function with silent range. A  $(\delta \circ \varphi)$ -gate  $g$  with  $k$  input variables  $\xi_1, \xi_2, \dots, \xi_k$  has weights  $w_1, w_2, \dots, w_k$  and a threshold  $t$ , where each of the weights and threshold are discretized by  $\delta$ . The output  $g(\xi_1, \xi_2, \dots, \xi_k)$  of  $g$  is then defined as

$$g(\xi_1, \xi_2, \dots, \xi_k) = \delta \circ \varphi \left( \sum_{i=1}^k w_i \xi_i - t \right).$$

A  $(\delta \circ \varphi)$ -circuit is a combinatorial circuit consisting of  $(\delta \circ \varphi)$ -gates except that the top gate  $g^{\text{clf}}$  is a threshold gate, that is, a linear classifier. We define size and depth of a  $(\delta \circ \varphi)$ -circuit same as the ones for a threshold circuit. We define energy  $e$  of a  $(\delta \circ \varphi)$ -circuit as the maximum number of gates outputting non-zero values in the circuit:

$$e = \max_{(\mathbf{a}, \mathbf{b}) \in \{0,1\}^{2n}} \sum_{g \in G} \llbracket g(\mathbf{a}, \mathbf{b}) \neq 0 \rrbracket$$

where  $\llbracket P \rrbracket$  for a statement  $P$  denote a notation of the function which outputs one if  $P$  is true, and zero otherwise. We define *weight*  $w$  of  $C$  as  $w = 2^{2b}$ , where  $2b$  is the bitwidth possibly needed to represent a potential value invoked by a single input of a gate in  $C$ .

## 2.2 Communication Matrix and its Rank

Let  $Z \subseteq \{0,1\}^n$ . For a Boolean function  $f : Z \times Z \rightarrow \{0,1\}$ , we define a communication matrix  $M_f$  over  $Z$  as a  $|Z| \times |Z|$  matrix where each row and column are indexed by  $\mathbf{a} \in Z$  and  $\mathbf{b} \in Z$ , respectively, and each entry is defined as  $M_f(\mathbf{a}, \mathbf{b}) = f(\mathbf{a}, \mathbf{b})$ . We denote by  $rk(M_f)$  the rank of  $M_f$  over  $\mathbb{F}_2$ . If a circuit  $C$  computes  $f$ , we may write  $M_C$  instead of  $M_f$ . If a Boolean function  $f$  does not have an obvious separation of the input variables to  $\mathbf{x}$  and  $\mathbf{y}$ , we may assume a separation so that  $rk(M_f)$  is maximized.

Let  $k$  and  $n$  be natural numbers such that  $k \leq n$ . Let

$$Z_k = \{\mathbf{a} \in \{0,1\}^n \mid \text{The number of ones in } \mathbf{a} \text{ is at most } k\}.$$

A  $k$ -disjointness function  $\text{DISJ}_{n,k}$  over  $Z_k$  is defines as follows:

$$\text{DISJ}_{n,k}(\mathbf{x}, \mathbf{y}) = \bigwedge_{i=1}^n \overline{x_i} \vee \overline{y_i}$$

where the input assignments are chosen from  $Z_k$ . The book [14] contains a simple proof showing that  $\text{DISJ}_{n,k}$  has full rank [14].

► **Theorem 4.**  $rk(M_{\text{DISJ}_{n,k}}) = \sum_{i=0}^k \binom{n}{i}$ . In particular,  $rk(M_{\text{DISJ}_{n,n}}) = 2^n$ .

$\text{CD}_n$  is the complement of  $\text{DISJ}_{n,n}$ . We can obtain the same bound for  $\text{CD}_n$ , as follows:

► **Corollary 5.**  $rk(M_{\text{CD}_n}) = 2^n$ .

We also use well-known facts on the rank. Let  $A$  and  $B$  be two matrices of same dimensions. We denote by  $A + B$  the summation of  $A$  and  $B$ , and by  $A \circ B$  the Hadamard product of  $A$  and  $B$ .

► **Fact 1.** For two matrices  $A$  and  $B$  of same dimensions, we have

(i)  $rk(A + B) \leq rk(A) + rk(B)$ ;

(ii)  $rk(A \circ B) \leq rk(A) \cdot rk(B)$ .

### 3 Lower Bound for Threshold Circuits

In this section, we give the inequality relating the rank of the communication matrix to the size, depth, energy and weight.

► **Theorem 6** (Theorem 1 restated). Let  $s, d, e$  and  $w$  be integers satisfying  $2 \leq s, d, 10 \leq e, 1 \leq w$ . Suppose a threshold circuit  $C$  computes a Boolean function of  $2n$  variables, and has size  $s$ , depth  $d$ , energy  $e$ , and weight  $w$ . Then it holds that

$$\log(rk(M_C)) \leq ed(\log s + \log w + \log n).$$

We prove the theorem by showing that  $M_C$  is a sum of matrices each of which corresponds to an internal representation that arises in  $C$ . Since  $C$  has bounded energy, the number of internal representations is also bounded. We then show by the inclusion-exclusion principle that each matrix corresponding to an internal representation has bounded rank. Thus, Fact 1 implies the theorem.

**Proof.** Let  $C$  be a threshold circuit that computes a Boolean function of  $2n$  variables, and has size  $s$ , depth  $d$ , energy  $e$  and weight  $w$ . Let  $G$  be a set of the gates in  $C$ . For  $l \in [d]$ , let  $G_l$  be a set of the gates in  $l$ -th level of  $C$ . Without loss of generality, we assume that  $G_d = \{g^{\text{clf}}\}$ . We evaluate the rank of  $M_C$ , and prove that

$$rk(M_C) \leq \left(\frac{c \cdot s}{e-1}\right)^{e-1} \cdot \left(\left(\frac{c \cdot s}{e-1}\right)^{e-1} \cdot (2nw+1)^{e-1}\right)^{d-1} \cdot (2nw+1) \quad (3)$$

where  $c < 3$ . Equation (3) implies that

$$rk(M_C) \leq \left(\frac{c \cdot s}{e-1} \cdot (2nw+1)\right)^{(e-1)d} \leq (snw)^{ed},$$

where the last inequality holds if  $e \geq 10$ . Taking the logarithm of the inequality, we obtain the theorem.

Below we verify that Eq. (3) holds. Let  $\mathbf{P} = (P_1, P_2, \dots, P_d)$ , where  $P_l$  is a subset of  $G_l$  for each  $l \in [d]$ . Given an input  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ , we say that an *internal representation*  $\mathbf{P}$  arises for  $(\mathbf{a}, \mathbf{b})$  if, for every  $l \in [d]$ ,  $g(\mathbf{a}, \mathbf{b}) = 1$  for every  $g \in P_l$ , and  $g(\mathbf{a}, \mathbf{b}) = 0$  for every  $g \notin P_l$ . We denote by  $\mathbf{P}^*(\mathbf{a}, \mathbf{b})$  the internal representation that arises for  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ . We then define  $\mathcal{P}_1$  as a set of the internal representations that arise for  $(\mathbf{a}, \mathbf{b})$  such that  $g^{\text{clf}}(\mathbf{a}, \mathbf{b}) = 1$ :

$$\mathcal{P}_1 = \{\mathbf{P}^*(\mathbf{a}, \mathbf{b}) \mid g^{\text{clf}}(\mathbf{a}, \mathbf{b}) = 1\}.$$

**85:8 Exponential Lower Bounds for Threshold Circuits of Sub-Linear Depth and Energy**

Note that, for any  $\mathbf{P} = (P_1, P_2, \dots, P_d) \in \mathcal{P}_1$ , we have  $|P_1| + |P_2| + \dots + |P_{d-1}| \leq e - 1$  and  $|P_d| = 1$ . Thus a standard upper bound on a sum of binomial coefficients implies that

$$|\mathcal{P}_1| \leq \sum_{k=0}^{e-1} \binom{s}{k} \leq \left( \frac{c \cdot s}{e-1} \right)^{e-1}. \quad (4)$$

For each  $\mathbf{P} \in \mathcal{P}_1$ , let  $M_{\mathbf{P}}$  be a  $2^n \times 2^n$  matrix such that, for every  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ ,

$$M_{\mathbf{P}}(\mathbf{a}, \mathbf{b}) = \begin{cases} 1 & \text{if } \mathbf{P} = \mathbf{P}^*(\mathbf{a}, \mathbf{b}); \\ 0 & \text{if } \mathbf{P} \neq \mathbf{P}^*(\mathbf{a}, \mathbf{b}). \end{cases}$$

By the definitions of  $\mathcal{P}_1$  and  $M_{\mathbf{P}}$ , we have

$$M_C = \sum_{\mathbf{P} \in \mathcal{P}_1} M_{\mathbf{P}},$$

and hence Fact 1(i) implies that

$$rk(M_C) \leq \sum_{\mathbf{P} \in \mathcal{P}_1} rk(M_{\mathbf{P}}).$$

Thus Eq. (4) implies that

$$rk(M_C) \leq \left( \frac{c \cdot s}{e-1} \right)^{e-1} \cdot \max_{\mathbf{P} \in \mathcal{P}_1} rk(M_{\mathbf{P}}).$$

We complete the proof by showing that, for any  $\mathbf{P} \in \mathcal{P}_1(C)$ , it holds that

$$rk(M_{\mathbf{P}}) \leq \left( \left( \frac{c \cdot s}{e-1} \right)^{e-1} \cdot (2nw + 1)^{e-1} \right)^{d-1} \cdot (2nw + 1).$$

In the following argument, we consider an arbitrary fixed internal representation  $\mathbf{P} = (P_1, P_2, \dots, P_d)$  in  $\mathcal{P}_1$ . We call a gate a *threshold function* if the inputs of the gate consists of only  $\mathbf{x}$  and  $\mathbf{y}$ . For each  $g \in G$ , we denote by  $\tau[g, \mathbf{P}]$  a threshold function defined as

$$\tau[g, \mathbf{P}](\mathbf{x}, \mathbf{y}) = \text{sign} (p_g^x(\mathbf{x}) + p_g^y(\mathbf{y}) + t_g[\mathbf{P}]).$$

where  $t_g[\mathbf{P}]$  is a threshold of  $g$ , being assumed that the internal representation  $\mathbf{P}$  arises:

$$t_g[\mathbf{P}] = \sum_{l=1}^{\text{lev}(g)-1} \sum_{h \in P_l} w_{h,g} - t_g.$$

For each  $l \in [d]$ , we define a set  $T_l$  of threshold functions as  $T_l = \{\tau[g, \mathbf{P}] \mid g \in G_l\}$ . Since every gate in  $G_1$  is a threshold function,  $T_1$  is identical to  $G_1$ .

For any set  $T$  of threshold functions, we denote by  $M[T]$  a  $2^n \times 2^n$  matrix such that, for every  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ ,

$$M[T](\mathbf{a}, \mathbf{b}) = \begin{cases} 1 & \text{if } \forall \tau \in T, \tau(\mathbf{a}, \mathbf{b}) = 1; \\ 0 & \text{if } \exists \tau \in T, \tau(\mathbf{a}, \mathbf{b}) = 0. \end{cases}$$

It is well-known that the rank of  $M[T]$  is bounded [8, 9], as follows. We give a proof for completeness.

▷ **Claim 7.**  $rk(M[T]) \leq (2nw + 1)^{|T|}$ .

Proof. Let  $z = |T|$ , and  $\tau_1, \tau_2, \dots, \tau_z$  be an arbitrary order of threshold functions in  $T$ . For each  $k \in [z]$ , we define

$$R_k = \{p_{\tau_k}^x(\mathbf{a}) \mid \mathbf{a} \in \{0, 1\}^n\}.$$

Since a threshold function receives a value between  $-w$  and  $w$  from a single input, we have  $|R_k| \leq 2nw + 1$ . For  $\mathbf{r} = (r_1, r_2, \dots, r_z) \in R_1 \times R_2 \times \dots \times R_z$ , we define  $R(\mathbf{r}) = X(\mathbf{r}) \times Y(\mathbf{r})$  as a combinatorial rectangle where

$$X(\mathbf{r}) = \{\mathbf{x} \mid \forall k \in [z], p_{\tau_k}(\mathbf{x}) = r_k\}$$

and

$$Y(\mathbf{r}) = \{\mathbf{y} \mid \forall k \in [z], t_{\tau_k} \leq r_k + p_{\tau_k}^y(\mathbf{y})\}.$$

Clearly, all the rectangles are disjoint, and hence  $M[T]$  can be expressed as a sum of rank-1 matrices given by  $R(\mathbf{r})$ 's taken over all the  $\mathbf{r}$ 's. Thus Fact 1(i) implies that its rank is at most  $|R_1 \times R_2 \times \dots \times R_z| \leq (2nw + 1)^z$ .  $\triangleleft$

For each  $l \in [d]$ , based on  $P_l$  in  $\mathbf{P}$ , we define a set  $Q_l$  of threshold functions as

$$Q_l = \{\tau[g, \mathbf{P}] \mid g \in P_l\} \subseteq T_l$$

and a family  $\mathcal{T}(Q_l)$  of sets  $T$  of threshold functions as

$$\mathcal{T}(Q_l) = \{T \subseteq T_l \mid Q_l \subseteq T \text{ and } |T| \leq e - 1\}.$$

Following the inclusion-exclusion principle, we define a  $2^n \times 2^n$  matrix

$$H[Q_l] = \sum_{T \in \mathcal{T}(Q_l)} (-1)^{|T| - |Q_l|} M[T].$$

We can show that  $M_{\mathbf{P}}$  is expressed as the Hadamard product of  $H[Q_1], H[Q_2], \dots, H[Q_d]$ :

$\triangleright$  Claim 8.  $M_{\mathbf{P}} = H[Q_1] \circ H[Q_2] \circ \dots \circ H[Q_d]$ .

Proof. Consider an arbitrary fixed assignment  $(\mathbf{a}, \mathbf{b}) \in \{0, 1\}^{2n}$ . We show that

$$H[Q_1](\mathbf{a}, \mathbf{b}) \circ H[Q_2](\mathbf{a}, \mathbf{b}) \circ \dots \circ H[Q_d](\mathbf{a}, \mathbf{b}) = 0,$$

if  $M_{\mathbf{P}}(\mathbf{a}, \mathbf{b}) = 0$ , and

$$H[Q_1](\mathbf{a}, \mathbf{b}) \circ H[Q_2](\mathbf{a}, \mathbf{b}) \circ \dots \circ H[Q_d](\mathbf{a}, \mathbf{b}) = 1,$$

if  $M_{\mathbf{P}}(\mathbf{a}, \mathbf{b}) = 1$ . We write  $\mathbf{P}^* = (P_1^*, P_2^*, \dots, P_d^*)$  to denote  $\mathbf{P}^*(\mathbf{a}, \mathbf{b})$  for a simpler notation.

Suppose  $M_{\mathbf{P}}(\mathbf{a}, \mathbf{b}) = 0$ . In this case, we have  $\mathbf{P} \neq \mathbf{P}^*$ , and hence there exists a level  $l \in [d]$  such that  $P_l \neq P_l^*$  while  $P_{l'} = P_{l'}^*$  for every  $l' \in [l - 1]$ . For such  $l$ , it holds that

$$\tau[g, \mathbf{P}^*](\mathbf{a}, \mathbf{b}) = \tau[g, \mathbf{P}](\mathbf{a}, \mathbf{b}) \tag{5}$$

for every  $g \in G_l$ . We show that  $H[Q_l](\mathbf{a}, \mathbf{b}) = 0$  by considering two cases:  $P_l \setminus P_l^* \neq \emptyset$  and  $P_l \subset P_l^*$ .

Consider the case where  $P_l \setminus P_l^* \neq \emptyset$ , then there exists  $g \in P_l \setminus P_l^*$ . Since  $g \notin P_l^*$ , we have  $\tau[g, \mathbf{P}^*](\mathbf{a}, \mathbf{b}) = 0$ . Thus, Eq. (5) implies that  $\tau[g, \mathbf{P}](\mathbf{a}, \mathbf{b}) = 0$ , and hence  $M[T](\mathbf{a}, \mathbf{b}) = 0$  for every  $T$  such that  $Q_l \subseteq T$ . Therefore, for every  $T \in \mathcal{T}(Q_l)$ , we have  $M[T](\mathbf{a}, \mathbf{b}) = 0$ , and hence

$$H[Q_l](\mathbf{a}, \mathbf{b}) = \sum_{T \in \mathcal{T}(Q_l)} M[T](\mathbf{a}, \mathbf{b}) = 0.$$

## 85:10 Exponential Lower Bounds for Threshold Circuits of Sub-Linear Depth and Energy

Consider the other case where  $P_l \subset P_l^*$ . Let  $Q_l^* = \{\tau[g, \mathbf{P}^*] \mid g \in P_l^*\}$ . Equation (5) implies that  $M[T](\mathbf{a}, \mathbf{b}) = 1$  if  $T$  satisfies  $Q_l \subseteq T \subseteq Q_l^*$ , and  $M[T](\mathbf{a}, \mathbf{b}) = 0$ , otherwise. Thus,

$$H[Q_l](\mathbf{a}, \mathbf{b}) = \sum_{T \in \mathcal{T}(Q_l)} (-1)^{|T| - |Q_l|} M[T] = \sum_{Q_l \subseteq T \subseteq Q_l^*} (-1)^{|T| - |Q_l|}$$

Therefore, by the binomial theorem,

$$H[Q_l](\mathbf{a}, \mathbf{b}) = \sum_{k=0}^{|Q_l^*| - |Q_l|} \binom{|Q_l^*| - |Q_l|}{k} (-1)^k = (1 - 1)^{|Q_l^*| - |Q_l|} = 0.$$

Suppose  $M_{\mathbf{P}}(\mathbf{a}, \mathbf{b}) = 1$ . In this case, we have  $\mathbf{P} = \mathbf{P}^*$ . Thus, for every  $l \in [d]$ , Eq. (5) implies that  $M[T](\mathbf{a}, \mathbf{b}) = 1$  if  $T = Q_l$ , and  $M[T](\mathbf{a}, \mathbf{b}) = 0$ , otherwise. Therefore,

$$H[Q_l](\mathbf{a}, \mathbf{b}) = \sum_{T \in \mathcal{T}(Q_l)} (-1)^{|T| - |Q_l|} M[T](\mathbf{a}, \mathbf{b}) = (-1)^{|Q_l| - |Q_l|} = 1.$$

Consequently,  $H[Q_1](\mathbf{a}, \mathbf{b}) \circ H[Q_2](\mathbf{a}, \mathbf{b}) \circ \cdots \circ H[Q_d](\mathbf{a}, \mathbf{b}) = 1$ , as desired.  $\triangleleft$

We finally evaluate  $rk(M_{\mathbf{P}})$ . Claim 8 and Fact 1(ii) imply that

$$rk(M_{\mathbf{P}}) = rk(H[Q_1] \circ H[Q_2] \circ \cdots \circ H[Q_d]) \leq \prod_{l=1}^d rk(H[Q_l]). \quad (6)$$

Since

$$|\mathcal{T}(Q_l)| \leq \left( \frac{c \cdot s}{e - 1} \right)^{e-1}$$

Fact 1(i) and Claim 7 imply that

$$\begin{aligned} rk(H[Q_l]) &\leq \sum_{T \in \mathcal{T}(Q_l)} rk(M[T]) \\ &\leq \left( \frac{c \cdot s}{e - 1} \right)^{e-1} \cdot (2nw + 1)^{e-1} \end{aligned} \quad (7)$$

for every  $l \in [d - 1]$ , and

$$rk(H[Q_d]) \leq 2nw + 1. \quad (8)$$

Equations (6)-(8) imply that

$$rk(M_{\mathbf{P}}) \leq \left( \left( \frac{c \cdot s}{e - 1} \right)^{e-1} \cdot (2nw + 1)^{e-1} \right)^{d-1} \cdot (2nw + 1)$$

as desired. We thus have verified Eq. (3).  $\blacktriangleleft$

Combining Corollary 5 and Theorem 6, we obtain the following corollary:

**► Corollary 9.** *Let  $s, d, e$  and  $w$  be integers satisfying  $2 \leq s, d, 10 \leq e, 1 \leq w$ . Suppose a threshold circuit  $C$  of size  $s$ , depth  $d$ , energy  $e$ , and weight  $w$  computes  $\text{CD}_n$ . Then it holds that*

$$n \leq ed(\log s + \log w + \log n).$$

*Equivalently, we have  $2^{n/(ed)}/(nw) \leq s$ .*

Theorem 6 implies lower bounds for other Boolean functions with linear rank. For example, consider another Boolean function  $\text{EQ}_n$  asking if  $\mathbf{x} = \mathbf{y}$ :

$$\text{EQ}_n(\mathbf{x}, \mathbf{y}) = \bigwedge_{i=1}^n \overline{x_i \oplus y_i}$$

Since  $M_{\text{EQ}_n}$  is the identity matrix with full rank, we have the same lower bound.

► **Corollary 10.** *Let  $s, d, e$  and  $w$  be integers satisfying  $2 \leq s, d, 10 \leq e, 1 \leq w$ . Suppose a threshold circuit  $C$  of size  $s$ , depth  $d$ , energy  $e$ , and weight  $w$  computes  $\text{EQ}_n$ . Then it holds that*

$$n \leq ed(\log s + \log w + \log n).$$

Equivalently, we have  $2^{n/(ed)}/(nw) \leq s$ .

## 4 Tightness of the Lower Bound

In this section, we show that the lower bound given in Theorem 6 is tight if the depth and energy are small.

### 4.1 Definitions

Let  $z$  be a positive integer, and  $f$  be a Boolean function of  $2n$  variables. We say that  $f$  is  $z$ -piecewise with  $f_1, f_2, \dots, f_z$  if the following conditions are satisfied: Let

$$B_j = \{i \in [n] \mid x_i \text{ or } y_i \text{ are fed into } f_j\},$$

then

- (i)  $B_1, B_2, \dots, B_z$  compose a partition of  $[n]$ ;
- (ii)  $|B_j| \leq \lceil n/z \rceil$  for every  $j \in [z]$ ;
- (iii)

$$f(\mathbf{x}, \mathbf{y}) = \bigvee_{j=1}^z f_j(\mathbf{x}, \mathbf{y}) \quad \text{or} \quad f(\mathbf{x}, \mathbf{y}) = \overline{\bigvee_{j=1}^z f_j(\mathbf{x}, \mathbf{y})}.$$

We say that a set of threshold gates sharing input variables is a neural set, and a neural set is selective if at most one of the gates in the set outputs one for any input assignment. A selective neural set  $S$  computes a Boolean function  $f$  if for every assignment in  $f^{-1}(0)$ , no gates in  $S$  outputs one, while for every assignment in  $f^{-1}(1)$ , exactly one gate in  $S$  outputs one. We define the size and weight of  $S$  as  $|S|$  and  $\max_{g \in S} w_g$ , respectively.

By a DNF-like construction, we can obtain a selective neural set of exponential size that computes  $f$  for any Boolean function  $f$ .

► **Theorem 11.** *For any Boolean function  $f$  of  $n$  variables, there exists a selective neural set of size  $2^n$  and weight one that computes  $f$ .*

### 4.2 Upper Bounds

The following proposition shows that we can construct threshold circuits of small energy for piecewise functions.

## 85:12 Exponential Lower Bounds for Threshold Circuits of Sub-Linear Depth and Energy

► **Lemma 12.** *Let  $e$  and  $d$  be integers satisfying  $2 \leq e$  and  $2 \leq d$ , and  $z$  be an integer. Suppose  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  is a  $z$ -piecewise function with  $f_1, f_2, \dots, f_z$ . If  $f_j$  is computable by a selective neural set of size at most  $s'$  and weight  $w'$  for every  $j \in [z]$ ,  $f$  is computable by a threshold circuit of size*

$$s \leq z \cdot s' + 1,$$

*depth  $d$ , energy  $e$  and weight*

$$w \leq \frac{2n}{z} \cdot w'.$$

Clearly,  $\text{CD}_n$  is a piecewise function, and so the lemma gives our upper bound for  $\text{CD}_n$ .

► **Theorem 13** (Theorem 2 restated). *For any integers  $e$  and  $d$  such that  $2 \leq e$  and  $2 \leq d$ ,  $\text{CD}_n$  is computable by a threshold circuit of size*

$$s \leq (e - 1)(d - 1) \cdot 2^{\frac{n}{(e-1)(d-1)}}.$$

*depth  $d$ , energy  $e$  and weight*

$$w \leq \left( \frac{n}{(e - 1)(d - 1)} \right)^2.$$

We can also obtain a similar proposition for  $\text{EQ}_n$ .

► **Theorem 14.** *For any integers  $e$  and  $d$  such that  $2 \leq e$  and  $2 \leq d$ ,  $\text{EQ}_n$  is computable by a threshold circuit of size*

$$s \leq (e - 1)(d - 1) \cdot 2^{\frac{2n}{(e-1)(d-1)}}.$$

*depth  $d$ , energy  $e$  and weight*

$$w \leq \frac{n}{(e - 1)(d - 1)}.$$

### 5 Simulating Discretized Circuits

In this section, we show that any discretized circuit can be simulated using a threshold circuit with a moderate increase in size, depth, energy, and weight. Thus, a similar inequality holds for discretized circuits, as follows.

► **Theorem 15.** *Let  $\delta$  be a discretizer and  $\varphi$  be an activation function such that  $\delta \circ \varphi$  has a silent range. If a  $(\delta \circ \varphi)$ -circuit  $C$  of size  $s$ , depth  $d$ , energy  $e$ , and weight  $w$  computes a Boolean function  $f$ , then it holds that*

$$\log(\text{rk}(M_C)) = O(ed(\log s + \log w + \log n)^3).$$

We prove the theorem by showing that, given a  $(\delta \circ \varphi)$ -circuit  $C$ , we can safely replace any  $(\delta \circ \varphi)$ -gate  $g$  in  $C$  by a set of threshold gates that simulate  $g$ . Our simulation is based on a binary search of the potentials of a discretized gate, and employ a conversion technique from a linear decision tree to a threshold circuit given in [42]. We omit our proof of the theorem due to the page limitation.

## 6 Conclusions

In this paper, we prove that a threshold circuit is able to compute only a Boolean function of which communication matrix has rank bounded by a product of logarithmic factors of size and weight, and linear factors of depth and energy. This bound implies that any threshold circuit of sub-linear depth, sub-linear energy and sub-exponential weight needs exponential size to compute  $CD_n$ ,  $EQ_n$ , and the Inner-Product function. We show that the bounds are tight up to a constant factor. We also prove that a similar bound holds for discretized circuits. Thus, increasing depth could help a neural network to acquire sparse activity. This observation may shed some light on the reason for the success of deep learning.

---

### References

- 1 Kazuyuki Amano. On the size of depth-two threshold circuits for the Inner Product mod 2 function. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications*, pages 235–247, Cham, 2020. Springer International Publishing.
- 2 Kazuyuki Amano and Akira Maruoka. On the complexity of depth-2 circuits with threshold gates. In *Proc. of the 30th international conference on Mathematical Foundations of Computer Science*, pages 107–118, 2005.
- 3 Ruiwen Chen, Rahul Santhanam, and Srikanth Srinivasan. Average-case lower bounds and satisfiability algorithms for small threshold circuits. *Theory of Computing*, 14(9):1–55, 2018.
- 4 Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, pages 3123–3131, Cambridge, MA, USA, 2015. MIT Press.
- 5 James J. DiCarlo and David D. Cox. Untangling invariant object recognition. *Trends in Cognitive Sciences*, 11(8):333–341, 2007. doi:10.1016/j.tics.2007.06.010.
- 6 Krishnamoorthy Dinesh, Samir Otiv, and Jayalal Sarma. New bounds for energy complexity of boolean functions. *Theoretical Computer Science*, 845:59–75, 2020.
- 7 Peter Földiák. Sparse coding in the primate cortex. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1064–1068. MIT Press, 2003.
- 8 Jürgen Forster, Matthias Krause, Satyanarayana V. Lokam, Rustam Mubarakzjanov, Niels Schmitt, and Hans Ulrich Simon. Relations between communication complexity, linear arrangements, and computational complexity. In *Proc. of the 21st International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 171–182, 2001.
- 9 András Hajnal, Wolfgang Maass, Pavel Pudlák, Máriaó Szegedy, and György Turán. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46:129–154, 1993.
- 10 Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, 1991.
- 11 Yunlong He, Koray Kavukcuoglu, Yun Wang, Arthur Szlam, and Yanjun Qi. Unsupervised feature learning by deep sparse coding. In *Proc. of SIAM International Conference on Data Mining*, pages 902–910, 2014.
- 12 Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018. URL: <http://jmlr.org/papers/v18/16-456.html>.
- 13 Russell Impagliazzo, Ramamohan Paturi, and Michael E. Saks. Size-depth tradeoffs for threshold circuits. *SIAM Journal on Computing*, 26(3):693–707, 1997.
- 14 Stasys Jukna. *Extremal Combinatorics with Applications in Computer Science*. Springer-Verlag Berlin Heidelberg, 2011.

- 15 D. M. Kane and R. Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In *Proc. of the 48th Annual ACM Symposium on Theory of Computing*, pages 633–643, 2016.
- 16 O. M. Kasim-zade. On a measure of active circuits of functional elements (russian). *Mathematical problems in cybernetics “Nauka”*, 4:218?–228, 1992.
- 17 Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Proc. of the 19th Advances in Neural Information Processing Systems*, pages 801–808, 2006.
- 18 Peter Lennie. The cost of cortical computation. *Current Biology*, 13:493–497, 2003.
- 19 Nancy Lynch and Cameron Musco. *A Basic Compositional Model for Spiking Neural Networks*, pages 403–449. Springer Nature Switzerland, 2022.
- 20 Wolfgang Maass, Georg. Schnitger, and Eduardo D. Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *Proc. of 32nd Annual Symposium of Foundations of Computer Science*, pages 767–776, 1991. doi:10.1109/SFCS.1991.185447.
- 21 Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.
- 22 Hiroki Maniwa, Takayuki Oki, Akira Suzuki, Kei, and Xiao Zhou. Computational power of threshold circuits of energy at most two. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E101.A(9):1431–1439, 2018.
- 23 Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- 24 Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1988.
- 25 Andrew Ng. Sparse autoencoder. CS294A Lecture notes, 2011.
- 26 Noam Nisan. The communication complexity of threshold gates. *Proceeding of “Combinatorics, Paul Erdős is Eighty”*, pages 301–315, 1993.
- 27 Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, 14(4):481–487, 2004.
- 28 Ian Parberry. *Circuit Complexity and Neural Networks*. MIT Press, 1994.
- 29 Thomas Pfeil, Tobias Potjans, Sven Schrader, Wiebke Potjans, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Is a 4-bit synaptic weight resolution enough? - constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Frontiers in Neuroscience*, 6:90, 2012. doi:10.3389/fnins.2012.00090.
- 30 Alexander A. Razborov and Alexander A. Sherstov. The sign-rank of  $AC^0$ . *SIAM Journal on Computing*, 39(5):1833–1855, 2010.
- 31 Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
- 32 Alexander A. Sherstov. Separating  $AC^0$  from depth-2 majority circuits. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC ’07, page 294?301, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250790.1250834.
- 33 Janio Carlos Nascimento Silva and Uéverton S. Souza. Computing the best-case energy complexity of satisfying assignments in monotone circuits. *Theoretical Computer Science*, 932:41–55, 2022.
- 34 K. Y. Siu and J. Bruck. On the power of threshold circuits with small weights. *SIAM Journal on Discrete Mathematics*, 4(3):423–435, 1991.
- 35 Kai-Yeung Siu, Vwani Roychowdhury, and Thomas Kailath. *Discrete Neural Computation: A Theoretical Foundation*. Prentice Hall, 1995.
- 36 Kai-Yeung Siu and Vwani P. Roychowdhury. On optimal depth threshold circuits for multiplication and related problems. *SIAM Journal on Discrete Mathematics*, 7(2):284–292, 1994.
- 37 Xiaoming Sun, Yuan Sun, Kewen Wu, and Zhiyu Xia. On the relationship between energy complexity and other boolean function measures. In *Proc. of the 25th International Computing and Combinatorics Conference*, pages 516–528, 2019.

- 38 Akira Suzuki, Kei Uchizawa, and Xiao Zhou. Energy-efficient threshold circuits computing MOD functions. In *Proc. of the 17th Computing: the Australasian Theory Symposium*, pages 105–110, 2011.
- 39 Akira Suzuki, Kei Uchizawa, and Xiao Zhou. Energy-efficient threshold circuits computing MOD functions. *International Journal of Foundations of Computer Science*, 24(1):15–29, 2013.
- 40 Kei Uchizawa. Lower bounds for threshold circuits of bounded energy. *Interdisciplinary Information Sciences*, 20(1):27–50, 2014.
- 41 Kei Uchizawa. Size, Depth and Energy of Threshold Circuits Computing Parity Function. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi : 10.4230/LIPIcs.ISAAC.2020.54.
- 42 Kei Uchizawa, Rodney Douglas, and Wolfgang Maass. On the computational power of threshold circuits with sparse activity. *Neural Computation*, 18(12):2994–3008, 2008.
- 43 Kei Uchizawa and Eiji Takimoto. Exponential lower bounds on the size of constant-depth threshold circuits with small energy complexity. *Theoretical Computer Science*, 407(1–3):474–487, 2008.
- 44 Kei Uchizawa, Eiji Takimoto, and Takao Nishizeki. Size-energy tradeoffs of unate circuits computing symmetric Boolean functions. *Theoretical Computer Science*, 412:773–782, 2011.
- 45 M. N. Vaintsvaig. On the power of networks of functional elements (Russian). *Doklady Akademii Nauk*, 139(2):320–323, 1961.



# Exact and Approximation Algorithms for Routing a Convoy Through a Graph

Martijn van Ee ✉

Netherlands Defence Academy, Den Helder, The Netherlands

Tim Oosterwijk ✉

Vrije Universiteit Amsterdam, The Netherlands

René Sitters ✉

Vrije Universiteit Amsterdam, The Netherlands

Andreas Wiese ✉

Technische Universität München, Germany

---

## Abstract

We study routing problems of a convoy in a graph, generalizing the shortest path problem (SPP), the travelling salesperson problem (TSP), and the Chinese postman problem (CPP) which are all well-studied in the classical (non-convoy) setting. We assume that each edge in the graph has a length and a speed at which it can be traversed and that our convoy has a given length. While the convoy moves through the graph, parts of it can be located on different edges. For safety requirements, at all time the whole convoy needs to travel at the same speed which is dictated by the slowest edge on which currently a part of the convoy is located. For Convoy-SPP, we give a strongly polynomial time exact algorithm. For Convoy-TSP, we provide an  $O(\log n)$ -approximation algorithm and an  $O(1)$ -approximation algorithm for trees. Both results carry over to Convoy-CPP which – maybe surprisingly – we prove to be NP-hard in the convoy setting. This contrasts the non-convoy setting in which the problem is polynomial time solvable.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** approximation algorithms, convoy routing, shortest path problem, traveling salesperson problem

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.86

## 1 Introduction

A fundamental setting within combinatorial optimization is to compute a route within a given graph, e.g., for a car in a street network. Well-studied problems in this area include the shortest path problem (SPP), the traveling salesperson problem (TSP), and the Chinese postman problem (CPP). For a single car, they model very well the problems of finding a route from one point to another, finding a tour that visits all given cities, or finding a tour that traverses all given edges, respectively.

However, this changes when instead we have a convoy that consists of several vehicles. Different streets in a network may allow different speeds. Because of safety considerations, the vehicles in the convoy need to adhere a constant inter-vehicle distance. Therefore, all the vehicles in the convoy need to travel at the same speed at any point in time. Hence, the *whole* convoy must move at the speed of the vehicle that is currently on the *slowest* edge (among all vehicles), see Figure 1. Thus, the classical algorithms for shortest path, TSP, and CPP cannot be used to find the fastest route for a convoy for these respective settings and they do not even yield approximation algorithms with any non-trivial approximation guarantee. Even more, shortest path and CPP are solvable in polynomial time in the usual setting, but it is not clear whether this is also the case for a convoy. Therefore, in this paper we investigate shortest path, TSP, and CPP in the convoy setting, present the first algorithms for these problems, and settle their complexity.



© Martijn van Ee, Tim Oosterwijk, René Sitters, and Andreas Wiese;  
licensed under Creative Commons License CC-BY 4.0

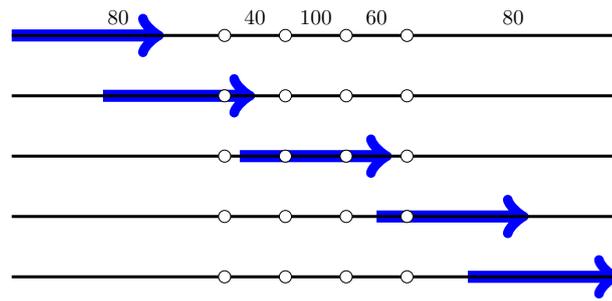
48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 86; pp. 86:1–86:15

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Five snapshots of a convoy moving along a path. The numbers along the edges denote the corresponding speeds. In the first snapshot, the convoy is entirely on an edge with speed 80, so it will move at speed 80. In the second snapshot, the front of the convoy is on an edge with speed 40, so the entire convoy will move at speed 40. In the third snapshot, the convoy still moves at speed 40. Note that in this example the convoy will never move at speed 100. In the fourth snapshot, the back of the convoy is on an edge with speed 60, so the convoy will move at speed 60. Finally, in the last snapshot, the convoy moves at speed 80.

Formally, we assume that we are given a directed graph  $G = (V, E)$ , where each edge  $e \in E$  has a length  $\ell(e) \geq 0$  and a speed  $q(e) > 0$ , and a length of the convoy  $L \geq 0$ . In Convoy-SPP we are given additionally two special vertices  $s, t \in V$  and the goal is to find a path from  $s$  to  $t$  that minimizes the time between the moment that the head of the convoy leaves  $s$  until the moment that the tail arrives in  $t$ . Figuratively, we assume that at the beginning all vehicles of the convoy are located in  $s$ . Therefore, when the head of the convoy leaves  $s$ , we assume that the parts of the convoy not having left  $s$  do not restrict the speed of the convoy. We make a symmetric assumption when the convoy enters  $t$ . In Convoy-TSP we are given a special vertex  $r \in V$  in which the convoy needs to start and end and we seek a tour for the convoy that visits every vertex  $v \in V$  at least once, minimizing the time between the moment that the head of the convoy leaves  $r$  until the moment that the tail arrives in  $r$  again (making the same assumptions when leaving and entering  $r$  as above for  $s$  and  $t$ ). Note that both problems can also be defined for undirected graphs, by simply replacing every undirected edge by two directed edges for both directions, with the same length and speed. Finally, in Convoy-CPP we assume that our graph  $G$  is undirected (as it is common when studying the Chinese postman problem in the classical setting, e.g., [3]) but again each edge  $e \in E$  has a length  $\ell(e) \geq 0$  and a speed  $q(e) > 0$ . Also, like in Convoy-TSP we are given a special vertex  $r \in V$  in which the convoy starts and ends. The goal is to find a tour that traverses every edge at least once, minimizing the time between the moment that the head of the convoy leaves  $r$  until the moment that the tail arrives in  $r$  again. In all problems, we assume without loss of generality that  $\min_{e \in E} q(e) = 1$ .

Convoy-SPP is the most basic routing problem for a convoy and it is motivated by any movement of a convoy from one point to another in a street network. The problem was introduced in [8] where the authors give an exponential time exact algorithm and argue that this “hints” that the problem is NP-hard. The Convoy-TSP problem arises for example when relief supplies need to be delivered to several locations, e.g., to several villages or cities, after a natural disaster or in unsecure territory. In these cases, moving in a convoy may be necessary so that the vehicles can support each other when roads are interrupted, or for protection. Similarly, Convoy-CPP arises when instead the convoy needs to visit all edges, e.g., because several villages are located on roads connecting the major cities. We are not aware of prior results for Convoy-TSP or Convoy-CPP. The setting of a convoy moving through a network can be considered as a variant of dynamic flows or flows over time, see [12].

## 1.1 Our contributions

We give an exact algorithm for Convoy-SPP that runs in strongly polynomial time. Note that this contrasts the argumentation mentioned above in [8]. A natural approach for Convoy-SPP would be to generalize algorithms for the usual shortest path problem, like Dijkstra's algorithm or Bellman-Ford. However, it is not clear how to do this. In both algorithms, we compute a value for each vertex  $v$  that indicates the distance of a shortest path from  $s$  to  $v$  (in the case of Bellman-Ford, we have one such value for each possible number of edges on this path). However, for continuing this path from  $v$  to  $t$ , it is crucial to know the speeds of the edges in which the convoy is located when reaching  $v$ . For example, there can be a long path from  $s$  to  $v$  such that the whole convoy is located on very fast edges when reaching  $v$ , and a short path such that the whole convoy is located on very slow edges when reaching  $v$ , and many other Pareto-optimal possibilities in between. We cannot afford to compute each of these possibilities. Instead, we use a very different approach. We guess the slowest edge  $(u, v)$  on the optimal path from  $s$  to  $t$ . The remaining problem splits into two independent subproblems: finding the optimal path from  $s$  to  $u$  and finding the optimal path from  $v$  to  $t$ . For each of these subproblems, we can assume that no edge slower than  $(u, v)$  is used. This leads to a polynomial number of possible subproblems which we solve via dynamic programming. Here, let  $n$  and  $m$  denote the number of vertices and edges in the graph, respectively.

► **Theorem 1.** *There is an algorithm for the Convoy-SPP with a running time of  $O(nm^2)$ .*

For usual TSP and asymmetric TSP there are polynomial time approximation algorithms known, e.g., [6, 14, 15]. However, it is not clear how to generalize them to Convoy-TSP, even if the graph were undirected. A natural attempt would be to define the cost of an edge  $e$  to be  $\ell(e)/q(e)$ , i.e., the time that it takes for an infinitesimally small convoy to traverse only this edge. However, the actual convoy of length  $L$  might be much slower than this on  $e$  if a part of it is located on a slower edge. In particular, even one single edge  $e$  can slow down the convoy for  $L/q(e)$  time units which might be much larger than  $\ell(e)/q(e)$ . On the other hand, we cannot add this value  $L/q(e)$  as a penalty to the cost of  $e$ , since if the convoy traverses many edges of speed  $q(e)$ , this penalty would overcount the slowdown by an arbitrarily large amount. Also, it is not obvious how to compute good lower bounds on the optimal solution: the strongest lower bound used in the known results for (asymmetric) TSP is the Held-Karp LP-relaxation [5] which does not take the convoy feature into account. It is not clear how to add this aspect to the LP.

Instead, we use a reduction to group-TSP (in the usual setting, not in the convoy setting). The intuition is the following. Suppose that  $v_1, v_2, \dots, v_n$  is the order in which the vertices are visited for the first time in the optimal solution. We identify that a key question is to determine for each vertex  $v_j$  the minimum speed at which the convoy travels while going from  $v_{j-1}$  to  $v_j$ . Therefore, in our reduction to group-TSP we introduce a copy of  $v_j$  for each possible speed and assign all these copies into the same group, i.e., only one of these copies needs to be visited. We introduce edges between these copies, corresponding to paths that use only edges with the respective minimum speeds, using our algorithm for Convoy-SPP as a subroutine. We show that our reduction loses only a constant factor in the approximation guarantee. By scaling the speeds to powers of 2, we ensure that there are intuitively only  $O(\log n)$  different speeds, and thus each group for some vertex  $v_j$  has at most  $O(\log n)$  copies of  $v_j$ . For such instances of group-TSP there is an  $O(\log n)$ -approximation [13] which we invoke as a subroutine.

We give a simple reduction from Convoy-CPP to Convoy-TSP which yields an  $O(\log n)$ -approximation algorithm for Convoy-CPP as well.

► **Theorem 2.** *There is a polynomial time  $O(\log n)$ -approximation algorithm for Convoy-TSP.*

► **Theorem 3.** *There is a polynomial time  $O(\log n)$ -approximation algorithm for Convoy-CPP.*

Maybe surprisingly, we prove that Convoy-CPP is NP-hard, which contrasts the fact that the usual Chinese postman problem is solvable in polynomial time [3].

► **Theorem 4.** *The Convoy-CPP is NP-hard, both for directed and undirected graphs, even when all edge lengths are equal to 1 and  $L = 2$ .*

Finally, we give an  $O(1)$ -approximation algorithm for Convoy-TSP in undirected trees. While for normal TSP in trees, a simple depth-first-search (DFS) is optimal, this is *not* the case for Convoy-TSP. In fact, DFS can yield a very bad solution, already on stars in which the edges have all length 1 and only two different speeds: assume that  $r$  is the center of the star, then the optimal solution visits first all slow edges and then all fast edges (or vice versa) while in a DFS solution this order can be arbitrary and the convoy could travel all the time with the low speed, which can be much slower than the optimal solution. While this particular example has an obvious optimal solution, it is not clear how to generalize it to arbitrary trees. Instead, we use a recursive approach. It is useful to imagine that our goal is to solve Convoy-CPP, which is equivalent to Convoy-TSP on trees, and hence we want to compute a tour that traverses all edges. We round the speeds to powers of 2 and form clusters of the edges with the lowest speed such that, intuitively, the edges in each cluster are close together and any two different clusters are far apart (at least  $L$  units). For each cluster we compute a DFS tour and argue that the optimal solution cannot be much faster when it traverses the edges of the cluster. Then, we remove all edges of the lowest speed and recurse on each connected component.

► **Theorem 5.** *There is a polynomial time  $O(1)$ -approximation algorithm for Convoy-TSP and Convoy-CPP on undirected trees.*

We defer the proof of this theorem to the full version of the paper. Our  $O(1)$ -approximation for trees is non-trivial and, although the analysis leaves room for a modest improvement of the constant 13, getting a ratio close to 1 needs a substantially different approach. In fact, an exact algorithm may still be possible. Also, it remains an open problem whether an  $O(1)$ -approximation for Convoy-TSP in arbitrary graphs exists.

## 1.2 Related work

The authors of [8] actually refer to our Convoy-SPP as the convoy quickest path problem. This is because there are some similarities between Convoy-SPP and the quickest path problem (QPP) [7]. In the QPP, we are given a directed graph  $G = (V, E)$ , where each edge  $e \in E$  has a time value  $d(e) \geq 0$ , and a capacity  $c(e) \geq 0$ . The capacity is an upper bound on the number of units per time unit that can pass through an edge. Here, we are also given two special vertices  $s, t \in V$ , and a length of a transmission  $\sigma \geq 0$ . The goal in QPP is to find a path from  $s$  to  $t$  minimizing the total time spent on the path plus the delay caused by the edge capacities. Note that this is different from our model in which the edges have speeds. Several polynomial time exact algorithms have been developed for QPP, we refer to [10] for a survey.

Another related problem is the convoy movement problem as studied in [1]. It is concerned with routing *multiple* convoys simultaneously such that there is no moment in time in which two convoys cross paths. Also in the variable speed convoy movement problem, studied in [11], the goal is to route several convoys to their respective destinations such that their paths do not cross.

### 1.3 Outline

The rest of the paper is organized as follows. In Section 2 we will present a dynamic programming algorithm that solves Convoy-SPP to optimality in polynomial time. Then in Section 3 we consider Convoy-TSP on arbitrary directed graphs, and we give an  $O(\log n)$ -approximation, where  $n$  is the number of vertices. In Section 4, we will show that Convoy-CPP is NP-hard, and show how the results for Convoy-TSP can be used to obtain similar results for Convoy-CPP.

## 2 Convoy-Shortest Path Problem

In this section we present an algorithm for Convoy-SPP with a running time of  $O(nm^2)$ , i.e., we prove Theorem 1. Without loss of generality we assume that the edges are indexed in non-increasing order of their speed, i.e., such that  $q(e_1) \geq q(e_2) \geq \dots \geq q(e_m) = 1$ . First, we show that we can restrict ourselves to instances of the form described in the statement of the following lemma. This will simplify our algorithm since it eliminates certain special cases, e.g., when the distance between  $s$  and  $t$  is shorter than  $L$ .

► **Lemma 6.** *Without loss of generality we can assume that  $s$  has exactly one outgoing edge  $e$  with  $q(e) = 1$  and  $\ell(e) = L$ , and  $t$  has exactly one incoming edge  $e'$  with  $q(e') = 1$  and  $\ell(e') = L$ .*

**Proof.** Consider an instance  $I$  of the convoy-shortest path problem on a graph  $G = (V, E)$  with speeds  $q(e_1) \geq \dots \geq q(e_m) = 1$ . Create an instance  $I'$  on the graph  $G' = (V', E')$  with  $V' = V \cup \{s', s'', t', t''\}$ , where  $E'$  includes all edges of  $E$ , but all edges connected to  $s$  and  $t$  are now connected to  $s''$  and  $t''$  respectively. We also add the new edges  $(s, s')$ ,  $(s', s'')$ ,  $(t'', t')$  and  $(t', t)$  to  $E'$  with speeds  $q((s, s')) = q((t', t)) = 1$  and  $q((s', s'')) = q((t'', t')) = q(e_1)$ , all of length  $L$ .

Any path in  $I$  can be transformed into a corresponding path in  $I'$  in which the convoy first traverses the edges  $(s, s')$  and  $(s', s'')$ , then takes the corresponding edge from  $s''$  followed by the same edges as in  $I$ , and finishes its path with the corresponding edge to  $t''$  and finally edges  $(t'', t')$  and  $(t', t)$ . The time of the path in  $I'$  is exactly  $4L$  longer than the time of the path in  $I$ . We can transform any path in  $I'$  similarly into a corresponding path in  $I$  that is  $4L$  shorter.

Since the solution values in  $I$  and  $I'$  differ only by a global constant that is independent of the respective solution, we can make this transformation for any instance  $I$  without loss of generality, solve the solution on the modified instance  $I'$ , and transform its solution back to a solution for  $I$ . ◀

From now on we denote with  $I'$  a given instance that satisfies the properties of Lemma 6. Our algorithm is a dynamic program (DP) that uses a divide and conquer strategy. Intuitively, we first guess the edge  $e_i = (v, w)$  with the largest index  $i$  on the path of the optimal solution; hence, no edge is slower on this path. This splits this path into three parts: a path from  $s$  to  $v$ , the edge  $(v, w)$ , and a path from  $w$  to  $t$ . Note that the first and the last part use only edges that have a smaller index than  $i$ , i.e., that are at least as fast as  $(v, w)$ . We recurse in order to find these two subpaths, i.e., we will ensure that there is a DP-cell for each of these subproblems. These two subproblems are independent, since while a part of the convoy is located on the edge  $(v, w)$ , that edge defines the speed of the convoy.

Formally, our DP-table has a cell  $CSP(u, (v, w))$  for combinations of a vertex  $u \in V$  and an edge  $e_i = (v, w) \in E$ . This cell encodes the following subproblem. First, place the convoy in the graph such that its tail is located on  $u$ , in any way you like. Then, we need to find a

path such that beginning with this starting position, the head of the convoy reaches  $v$  as fast as possible, while we are only allowed to use edges in the set  $E_{(v,w)} = \{e_1, \dots, e_{i-1}\}$  (i.e., edges with index lower than the index of  $(v, w)$  and which are hence at least as fast as  $(v, w)$ ). In this cell  $CSP(u, (v, w))$  we store the time the convoy needs to traverse this path; we also store the path itself in order to determine later the optimal path and not only its time. Note that the edges on which we place the convoy in the first step are part of the computed path but there is no cost involved in this placement. Also note that it is sufficient that the head of the convoy reaches  $v$  but it does not need to traverse the edge  $(v, w)$ . We include  $(v, w)$  in the description of the cell  $CSP(u, (v, w))$  because it determines  $E_{(v,w)}$ , the subset of the edges that the convoy is allowed to use in its path. We introduce a cell  $CSP(u, (v, w))$  for each combination of a vertex  $u \in V$  and an edge  $e_i = (v, w) \in E$  such that the shortest path from  $u$  to  $v$  using only edges in  $E_{(v,w)}$  has a length of more than  $L$ . We will see later that if the shortest path from  $u$  to  $v$  has length at most  $L$ , then we do not need to recurse on that subproblem but we can simply take the shortest path from  $u$  to  $v$  in  $E_{(v,w)}$ .

Suppose that we are given a  $CSP(u, (v, w))$ . Let  $e_{i'} = (v', w') \in E_{(v,w)}$  be the edge with largest index  $i'$  in the path  $P^*$  of the optimal solution corresponding to the cell  $CSP(u, (v, w))$ . Suppose that the part of  $P^*$  between  $u$  and  $v'$  has a length of at least  $L$  and that the same holds for the part of  $P^*$  between  $w'$  and  $v$ . Then the former part is the optimal solution to the DP-cell  $CSP(u, (v', w'))$  and after that the convoy needs  $(\ell(e_{i'}) + L)/q(e_{i'})$  time units for traversing  $e_{i'}$ . The last part of  $P^*$  would be the optimal solution to the DP-cell  $CSP(v, (w', v'))$  if the edges of  $G$  were all reversed, so if each edge  $(\hat{u}, \hat{v})$  were replaced by the edge  $(\hat{v}, \hat{u})$  with the same length and speed as  $(\hat{u}, \hat{v})$ . Therefore, we define a “reversed” auxiliary graph  $G^R = (V, E^R)$  with  $E^R = \{(v, u) \mid (u, v) \in E\}$  where every reversed edge  $e^R \in E^R$  has the same speed and length as its original  $e \in E$ . Also, we define the same DP-cells as above for the same subproblems for  $G^R$ , i.e., we define a cell  $CSP^R(\hat{u}, (\hat{w}, \hat{v}))$  for each combination of a vertex  $\hat{u} \in V$  and an edge  $(\hat{w}, \hat{v}) \in E^R$  (so  $(\hat{v}, \hat{w}) \in E$ ) such that the shortest path from  $\hat{u}$  to  $\hat{w}$  in  $G^R$  has a length of more than  $L$ . Thus, we can split  $P^*$  into the optimal solution for  $CSP(u, (v', w'))$ , the edge  $(v', w')$ , and the optimal solution to  $CSP^R(v, (w', v'))$ , and it takes the convoy a total time of  $CSP(u, (v', w')) + (\ell(e_{i'}) + L)/q(e_{i'}) + CSP^R(v, (w', v'))$  to traverse  $P^*$ .

If the part of  $P^*$  from  $u$  to  $v'$  has length at most  $L$ , then one can show that it coincides simply with the shortest path from  $u$  to  $v'$  using only edges in  $E_{(v',w')}$  (since the edge  $(v', w')$  determines the speed of the convoy anyway), and a similar statement holds for the part of  $P^*$  from  $w'$  to  $v$ . Thus, when we want to compute the entry of the cell  $CSP(u, (v, w))$ , we guess the edge  $e_{i'} = (v', w')$ , i.e., we try each edge in  $E_{(v,w)}$ , we compute the lengths of the shortest paths from  $u$  to  $v'$  and from  $w'$  to  $v$  using only edges in  $E_{(v',w')}$ , and depending on their lengths look up the DP-cells  $CSP(u, (v', w'))$  and  $CSP^R(v, (w', v'))$  or take these shortest paths directly.

This yields the following recursive formula, where for each combination of a vertex  $u$  and an edge  $(v, w) \in E$  we denote by  $SP(u, (v, w))$  the shortest path in  $G$  from  $u$  to  $v$  that uses only edges in  $E_{(v,w)}$ ; we define  $SP^R(u, (v, w))$  similarly for  $G^R$ .

$$CSP(u, (v, w)) = \min \left\{ \begin{array}{l} \min_{\substack{e \in E_{(v,w)}: \\ SP(u,e) > L, \\ SP^R(v,e^R) > L}} \left\{ CSP(u, e) + \frac{\ell(e) + L}{q(e)} + CSP^R(v, e^R) \right\}, \\ \min_{\substack{e \in E_{(v,w)}: \\ SP(u,e) \leq L, \\ SP^R(v,e^R) > L}} \left\{ \frac{SP(u, e) + \ell(e)}{q(e)} + CSP^R(v, e^R) \right\}, \end{array} \right. \quad (1)$$

$$\min_{\substack{e \in E(v,w): \\ SP(u,e) > L, \\ SP^R(v,e^R) \leq L}} \left\{ CSP(u,e) + \frac{\ell(e) + SP^R(v,e^R)}{q(e)} \right\},$$

$$\min_{\substack{e \in E(v,w): \\ SP(u,e) \leq L, \\ SP^R(v,e^R) \leq L}} \left\{ \frac{SP(u,e) + \ell(e) + SP^R(v,e^R) - L}{q(e)} \right\}.$$

Note that we need not define base cases for this dynamic program. All cells where both subpaths are shorter than  $L$  are computed using  $SP$ . In the first cell where we recurse on a previously computed cell  $C$  of the DP, the length of the path corresponding to  $C$  is larger than  $L$ , but cell  $C$  has already been computed at that point because its value will be attained because of the fourth term of the recursion using  $SP$ .

We could compute all shortest paths in the graph restricted to  $E_{(v,w)}$  for each  $(v,w) \in E$  with any shortest path algorithm, e.g., with Dijkstra's algorithm. However, in order to improve the running time slightly, we use another DP which is analogous to the DP above. We initialize  $SP(v,(v,w)) = 0$  for all  $(v,w) \in E$ ,  $SP^R(v,(v,w)) = 0$  for all  $(v,w) \in E^R$ , and we use the recurrence

$$SP(u,(v,w)) = \min_{e \in E(v,w)} \{ SP(u,e) + \ell(e) + SP^R(v,e^R) \},$$

for each cell  $SP(u,(v,w))$  with  $u \neq v$  and a symmetric recurrence for each entry  $SP^R(u,(v,w))$  for which  $u \neq v$ .

Finally, we output the path stored in  $CSP(s,(t',t))$ . Note that the time stored in that cell corresponds to the time that the convoy, starting with its tail in  $s$ , needs such that its head reaches  $t'$ . The convoy needs an additional  $L$  time units in order to leave  $s$ , an additional  $L$  time units to reach  $t$ , and an additional  $L$  time units to enter  $t$  completely. This yields our solution to the transformed instance  $I'$ , which we can transform back to a solution to the original instance  $I$  using Lemma 6.

► **Theorem 1.** *There is an algorithm for the Convoy-SPP with a running time of  $O(nm^2)$ .*

**Proof.** First we prove correctness and then we prove the running time.

Consider a cell  $CSP(u,(v,w))$  for which we want to compute the optimal solution, which is a path  $P$  from  $u$  to  $v$  that minimizes the convoy travel time. We guess the slowest edge in  $P$ , let this be  $e^* = (u^*,v^*)$ . Let  $P_1$  and  $P_2$  be the subpaths of  $P$  before and after  $e^*$  respectively (both excluding  $e^*$ ).

First consider the case where both  $P_1$  and  $P_2$  are strictly larger than  $L$ . Then we can subdivide the problem encoded in this cell of the DP by using edge  $e^*$ . The total time it takes for the convoy to travel along  $P$  is the time to travel along  $P_1$  plus the time to travel along  $e^*$  plus the time to travel along  $P_2$ . Since both  $P_1$  and  $P_2$  are larger than  $L$  and they do not contain this new bottleneck edge  $e^*$ , their travel time is equal to  $CSP(u,e^*)$  and  $CSP^R(v,e^*)$  respectively. Note that these terms indeed do not contain the travel time along  $e^*$ , as the first term is the travel time until the head of the convoy reaches  $e^*$  and the second term is the travel time from the point in time onward when the tail leaves  $e^*$ . Finally, the travel time along  $e^*$  that is missing is the time it takes for the convoy to completely traverse the edge  $e^*$  from head to tail. This equals  $(\ell(e^*) + L)/q(e^*)$ , since this bottleneck edge determines the speed of the convoy for all these time units. Together, this gives us the first term of Equation (1).

Now observe that if  $SP(u, e^*) \leq L$ , there exists a path from  $u$  to  $u^*$  such that the tail of the convoy has just left  $u$  and some part of the convoy has reached  $u^*$ . Thus, edge  $e^*$  is the edge that determines the speed of the convoy while it is moving along  $P_1$  and along  $e^*$ . Since it travels a distance of  $SP(u, e^*) + \ell(e^*)$  there at speed  $q(e^*)$ , this takes  $(SP(u, e^*) + \ell(e^*)) / q(e^*)$  time units. Then, if the second part of the path is longer than  $L$  (i.e., if  $SP^R(v, e^*) > L$ ),  $P_2$  is the path corresponding to the solution  $CSP^R(v, e^*)$  and we retrieve the second term of Equation (1).

Similarly, if  $SP^R(v, e^*) \leq L$ , then  $P_2 = SP^R(v, e^*)$ , and by the same arguments as above we see that the third term of Equation (1) is correct.

Finally, if both  $SP(u, e^*) \leq L$  and  $SP^R(v, e^*) \leq L$ , then this total path is traversed at speed  $e^*$ . The total length of the path that the convoy needs to cover between  $u$  and  $v$  is therefore  $SP(u, e^*) + \ell(e^*) + SP^R(v, e^*)$ . Since we measure the time it takes for the tail to leave  $u$  and the head to reach  $v$ , we need to subtract  $L$  from this distance to get the distance that the convoy traverses. Note that the distance  $SP(u, e^*) + \ell(e^*) + SP^R(v, e^*) - L$  must be positive because we only introduce this cell whenever  $SP(u, (v, w)) > L$ . Dividing by the speed  $q(e^*)$  at which the convoy traverses this distance, we obtain the last term of Equation (1). This finishes the proof of correctness of the algorithm.

Using these observations, the entry of the cell  $CSP(u, (v, w))$  can be computed using only previously computed entries of cells of shorter paths. There are  $O(nm)$  cells in each of the dynamic programs. In each computation there are  $O(m)$  terms that need to be compared for the minimum value, since we recurse on one of the edges of the path. Therefore, the total running time of each of the dynamic programs is bounded by  $O(nm^2)$ . ◀

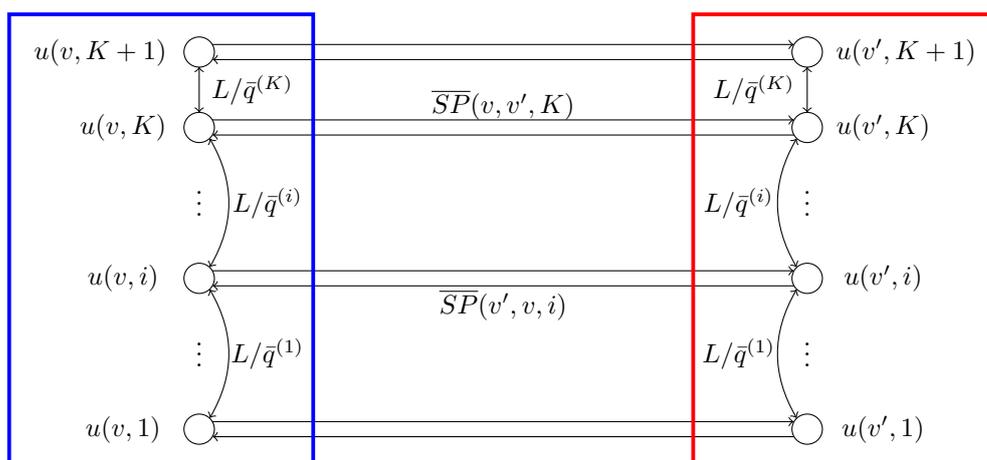
### 3 Convoy-TSP in general graphs

In this section we present our  $O(\log n)$ -approximation algorithm for Convoy-TSP in general graphs. First, we do some guesses and preprocessing in order to simplify our problem, while losing only a constant factor in our approximation guarantee.

► **Lemma 7.** *By losing a factor of 2 in the approximation ratio, we can assume that  $\ell(e) \geq L/nm$  for each edge  $e$ .*

**Proof.** Let  $\epsilon = L/nm$ . First, increase the length of any edge with length smaller than  $\epsilon$  to  $\epsilon$ . It follows that the optimal value of the modified instance cannot be smaller than the optimal value of the original instance. Now, consider the optimal solution of the original instance. Let  $e^*$  be the slowest edge with length less than  $\epsilon$  that is traversed by the optimal solution. We know that the optimal value of the original instance is at least  $L/q(e^*)$ . If we follow the same tour in the modified instance, we have to spend at most  $\epsilon/q(e^*)$  time additionally for each modified edge. This is true since staying longer on an edge will not decrease the speed of the convoy. In total, we spend at most  $nm\epsilon/q(e^*) = L/q(e^*)$  time units extra, so we lose at most a factor 2. ◀

From now on we assume that all edges have length at least  $L/nm$ . We say that an edge  $e$  is *short* if  $\ell(e) \leq 2L$  and *long* otherwise. Intuitively, the long edges are easy to deal with: if the convoy traverses a long edge  $e$ , then the whole convoy will be on  $e$  while traversing a distance of at least  $L$ , which takes at least  $L/q(e)$  time. Potentially, for some time only a part of the convoy is on  $e$  and then  $e$  slows the convoy down. However, this time is at most  $2L/q(e)$  which can be charged to the previous time of  $L/q(e)$ . In particular, if we had only long edges then we could reduce the problem to (normal) TSP by losing only a factor  $O(1)$  by simply giving each (long) edge length  $(\ell(e) + L)/q(e)$ .



■ **Figure 2** Illustration of constructed instance of group-TSP. Note that there is an edge between any pair of the form  $u(v, i)$  and  $u(v, j)$ . From the set of vertices enclosed by both the blue and red rectangle, at least one vertex has to be visited.

Thus, the core difficulty of the problem stems from the short edges. We want to modify the instance such that the short edges have only  $O(\log n)$  different speeds.

► **Lemma 8.** *By losing a factor of 6 in the approximation ratio, we will assume that the short edges have only  $O(\log n)$  different speeds and that they are all powers of 2.*

**Proof.** We guess the short edge with the lowest speed that is used in OPT. Let  $e^*$  be this edge. First, we remove each short edge that is slower than  $e^*$ . Then, if a short edge  $e$  is faster than  $n^2 m^2 \cdot q(e^*)$ , we round its speed down to  $n^2 m^2 \cdot q(e^*)$ . Since these edges are short, and any tour uses at most  $nm$  edges, this rounding will incur at most an additional

$$nm \cdot \frac{2L}{n^2 m^2 q(e^*)} = \frac{2L}{nmq(e^*)} \leq \frac{2\ell(e^*)}{q(e^*)} \leq 2\text{OPT}$$

time units, where the first inequality follows from Lemma 7. Hence, we lose at most a factor 3 in the approximation ratio. Finally, we round all speeds of the short edges down to powers of 2, which loses us another factor 2, to obtain  $O(\log n^2 m^2) = O(\log n)$  different speeds for the short edges. ◀

Let  $\bar{q}^{(1)} < \bar{q}^{(2)} < \dots < \bar{q}^{(K)}$  denote the different speeds of the short edges, and by Lemma 8 we have that  $K = O(\log n)$ . We guess the speed of the slowest short edge used in OPT and remove all slower short edges. Hence, we assume w.l.o.g. that OPT uses a short edge of speed  $\bar{q}^{(1)}$ .

We define now an instance of asymmetric group-TSP to which we reduce our given instance of Convoy-TSP. In group-TSP, the input consists of a directed graph  $G' = (V', E')$ , a weight  $w(e)$  for each  $e \in E'$  and a set of groups  $\mathcal{V}'_1, \dots, \mathcal{V}'_s \subseteq V'$ . Intuitively, the edge weights denote the lengths of the edges  $E'$ , but we call them weights here in order to distinguish them better from the lengths  $\ell(e)$  of the edges  $e \in E$ . The goal is to compute a minimum weight tour that visits at least one vertex from each group  $\mathcal{V}'_i$ .

For each input vertex  $v \in V$  we define a group of  $K + 1$  vertices  $\{u(v, 1), \dots, u(v, K + 1)\}$  (see Figure 2), i.e., the tour needs to visit at least one of these vertices. Intuitively, for each  $i \in \{1, \dots, K\}$  the vertex  $u(v, i)$  corresponds to arriving at the vertex  $v$  at speed  $\bar{q}^{(i)}$ , while the vertex  $u(v, K + 1)$  corresponds to arriving at  $v$  after traversing a long edge  $e$  (and then

the speed does not matter much since any slowdown due to  $e$  can be charged to the time the convoy spends on  $e$ ). For any pair of vertices  $v, v' \in V$  and each  $i \in \{1, \dots, K+1\}$  we introduce a directed edge  $e = (u(v, i), u(v', i))$  and we define the weight  $w(e)$  to be the time of the quickest path from  $v$  to  $v'$  such that

- at the beginning the head of the convoy is on  $v$  and the convoy travels at speed at most  $\bar{q}^{(i)}$  until the tail of the convoy has left  $v$ ,
- at the end the head of the convoy is at  $v'$ ,
- the convoy is allowed to use only short edges of speed at least  $\bar{q}^{(i)}$  and long edges (of any speed); if  $i = K+1$  it is only allowed to use long edges.

We denote by  $\overline{SP}(v, v', i)$  the corresponding path. We can compute  $\overline{SP}(v, v', i)$  with our algorithm from Section 2.

► **Lemma 9.** *For each  $v, v' \in V$  and each  $i \in \{1, \dots, K+1\}$  we can compute  $\overline{SP}(v, v', i)$  in time  $O(nm^2)$ .*

**Proof.** From graph  $G$ , create an auxiliary graph  $\bar{G}$  with a new vertex  $\bar{v}$  and an edge  $\bar{e} = (\bar{v}, v)$  with speed  $\bar{q}^{(i)}$  and length  $L$ . Moreover, delete all short edges from  $E'$  with a speed slower than  $\bar{q}^{(i)}$ . Indeed, the solution to that instance gives us exactly  $\overline{SP}(v, v', i)$  as requested, which can be solved using the algorithm from Theorem 1. ◀

Also, for each vertex  $v$  and any  $i, i' \in \{1, \dots, K+1\}$  we introduce the edge  $e = (u(v, i), u(v, i'))$ , and define its weight  $w(e) := L / \min\{\bar{q}^{(i)}, \bar{q}^{(i')}\}$ . Intuitively, this models that if the convoy traverses some edge of speed  $\bar{q}^{(i)}$  and after that edges of a higher speed  $\bar{q}^{(i')}$ , then the edge of speed  $\bar{q}^{(i)}$  slows down the convoy for  $L/\bar{q}^{(i)}$  more time units. To streamline our argumentation, we introduce this edge with this cost also if  $\bar{q}^{(i')} < \bar{q}^{(i)}$ .

We solve our instance of group-TSP with an adaptation of the algorithm in [13] which yields an  $O(K)$ -approximate tour  $T'$ . We translate  $T'$  to a tour  $T$  of our given instance of Convoy-TSP as follows. We define  $T$  by adding to it step by step the edges that the convoy needs to traverse. We know that  $T'$  must visit at least one vertex  $u(r, i)$  of the group corresponding to the initial vertex  $r \in V$  of the convoy. Starting in  $u(r, i)$ , we follow  $T'$ . Whenever  $T'$  uses an edge  $e = (u(v, i), u(v', i))$  for two vertices  $v, v'$  and an  $i \in \{1, \dots, K+1\}$  then we add to  $T$  the edges in  $\overline{SP}(v, v', i)$ . When the convoy travels along an edge  $e = (u(v, i), u(v, i'))$  for a vertex  $v$  and  $i, i' \in \{1, \dots, K+1\}$  we do not add edges to  $T$  (such an edge  $e$  ensures only that the time of our convoy in tour  $T$  is comparable to  $w(T')$ ).

► **Lemma 10.** *The convoy needs time  $O(w(T'))$  to traverse the tour  $T$ .*

**Proof.** Suppose that when the convoy traverses  $T$  it does this during the time interval  $I$ . Let  $I_1$  denote the union of all subintervals of  $I$  during which the speed of the convoy is determined by a long edge  $e$ . Consider one such edge  $e$ . Then there are two vertices  $v, v'$  and an  $i \in \{1, \dots, K+1\}$  such that  $(u(v, i), u(v', i)) \in T'$  and  $e \in \overline{SP}(v, v', i)$ . Thus, the time during  $I$  in which  $e$  determines the speed of the convoy is at most  $\frac{L+\ell(e)}{q(e)}$ , while  $e$  contributes at least  $\frac{\ell(e)}{q(e)} = \Omega\left(\frac{L+\ell(e)}{q(e)}\right)$  to  $w((u(v, i), u(v', i)))$ . Therefore,  $|I_1| \in O(w(T'))$ .

Let  $I_2 := I \setminus I_1$ , i.e., the union of the time intervals in  $I$  during which the speed of the convoy is determined by a short edge. We want to bound  $|I_2|$ . To this end, take  $T'$  and consider all connected components of edges of the form  $e = (u(v, i), u(v', i))$  for two vertices  $v, v'$  and an  $i \in \{1, \dots, K+1\}$ . Let  $C = \{e_1, e_2, \dots\}$  be such a component. Then there exists an  $i \in \{1, \dots, K+1\}$  such that for each edge  $e_j \in C$  there are two vertices  $v_j, v'_j \in V$  such that  $e_j = (u(v_j, i), u(v'_j, i))$  and the convoy traverses the path  $\overline{SP}(v_j, v'_j, i)$ . The total time that the convoy traverses the paths  $\{\overline{SP}(v_j, v'_j, i)\}_{j:e_j \in C}$  such that the speed of the convoy

is determined by a short edge of speed *at least*  $\bar{q}^{(i)}$  is bounded by  $\sum_{j:e_j \in C} w(e_j)$  (note that  $w(e_j)$  might over-estimate the time needed by the convoy to traverse the edges of  $\overline{SP}(v_j, v'_j, i)$  since we assume that the convoy travels at speed at most  $\bar{q}^{(i)}$  when leaving  $v_j$ ). It remains to bound the time that the convoy traverses the paths  $\{\overline{SP}(v_j, v'_j, i)\}_{j:e_j \in C}$  such that the speed of the convoy is determined by a short edge of some speed that is strictly *slower* than  $\bar{q}^{(i)}$ ; we denote by  $\hat{t}$  this amount of time. Let  $\bar{q}^{(i^*)}$  be the smallest such speed. In particular, this implies that  $\hat{t} \leq L/\bar{q}^{(i^*)}$ .

Let  $C'$  denote the connected component of the edges of the form  $e = (u(v, i), u(v, i'))$  for a vertex  $v \in V$  and  $i, i' \in \{1, \dots, K+1\}$  that is traversed in  $T'$  right before  $C$ . If  $\hat{t} > 0$ , then  $C'$  must contain an edge  $e^* = (u(v, i^*), u(v, \hat{i}))$  with  $\bar{q}^{(i^*)} < \bar{q}^{(\hat{i})}$  and thus  $w(e^*) = L/\bar{q}^{(i^*)} \geq \hat{t}$ . Hence, the total time that the convoy traverses  $C$  is bounded by  $w(e^*) + \sum_{j:e_j \in C} w(e_j)$ . Applying this argumentation to all components  $C$  yields that  $|I_2| \in O(w(T'))$  and thus  $|I| = |I_1| + |I_2| \in O(w(T'))$ . ◀

Abusing notation, we denote by OPT the time that it takes to traverse OPT. It remains to show that  $w(T') = O(\log n \cdot \text{OPT})$ . To this end, we show that we can construct a tour  $T^*$  for our instance of group-TSP for which  $w(T^*) = O(\text{OPT})$  holds. This implies that  $w(T') = O(K \cdot w(T^*)) = O(\log n \cdot \text{OPT})$ . Note that hence our reduction to group-TSP loses only a factor  $O(1)$  (and we lose the factor  $O(\log n)$  because we can solve the instance of group-TSP only approximately).

Based on OPT, we describe now how we construct  $T^*$  such that  $w(T^*) = O(\text{OPT})$ . Assume w.l.o.g. that  $V = \{v_1, \dots, v_n\}$  such that  $v_1 = r$  (i.e., the initial vertex) and for each  $j \in \{1, \dots, n-1\}$  we have that  $v_j$  is visited in OPT for the first time before  $v_{j+1}$  is visited in OPT for the first time (note that OPT might visit a vertex several times). For convenience, we define that  $v_{n+1} = v_1$ . For each  $j \in \{1, \dots, n\}$  we do the following: let  $P_j$  denote the path that the convoy takes between visiting  $v_j$  and  $v_{j+1}$  for the first time. If  $P_j$  contains a short edge, we add  $(u(v_j, i), u(v_{j+1}, i))$  to  $T$  where  $\bar{q}^{(i)}$  is the minimum speed at which the convoy travels in OPT while the head of the convoy is on an edge in  $P_j$ . If  $P_j$  does not contain a short edge, then we add  $(u(v_j, K+1), u(v_{j+1}, K+1))$  to  $T$ .

It can happen that for some vertex  $v_j$  we added two edges  $(u(v_{j-1}, i), u(v_j, i))$ ,  $(u(v_j, i'), u(v_{j+1}, i'))$  with  $i \neq i'$ . In this case we add the edge  $(u(v_j, i), u(v_j, i'))$ . Let  $T^*$  be the resulting tour.

▶ **Lemma 11.** *The group-TSP solution  $T^*$  has total weight  $w(T^*) = O(\text{OPT})$ .*

**Proof.** Suppose that we change OPT to a new tour OPT' by slowing down the convoy even though it would not be necessary. We do this such that OPT' can be traversed in at most time  $O(\text{OPT})$ . Let  $\ell(\text{OPT})$  denote the total length that the head of the convoy travels in OPT. Here, we assume w.l.o.g. that the head keeps traveling after reaching  $r$  for a distance of  $L$  using edges of unbounded speed. Now, consider the interval  $I = [0, \ell(\text{OPT})]$ . For each point  $d \in I$  denote by  $q_{\text{OPT}}(d)$  the speed that the convoy travels in OPT when the head of the convoy has already traveled a distance of  $d$ . Let  $I' = [a, b] \subseteq I$  denote a maximally large subinterval of  $I$  such that  $q_{\text{OPT}}(d) = \bar{q}^{(1)}$  for each  $d \in I'$ . Let  $a'$  denote the largest value  $x \leq a$  such that after travelling a distance of  $x$  in OPT the head of the convoy visits a vertex  $v \in V$  for the first time. We define a new speed function  $q'$  for the head of the convoy for which we define for each  $d \in [a', \min\{a' + L, b\}]$  that  $q'(d) := \bar{q}^{(1)}$ .

Similarly, let  $b'$  denote the smallest value  $y \geq b$  such that after travelling a distance of  $y$  in OPT the head of the convoy visits a vertex  $v \in V$  for the first time. We define  $q'$  such that for each  $d \in [\max\{b' - L, a\}, b']$  we define the speed  $q'(d) := \bar{q}^{(1)}$  for the head of the convoy. We define that the interval  $[a', b']$  is *clean*. We do this operation for each interval

$I' \subseteq I$  such that  $q_{\text{OPT}}(d) = \bar{q}^{(1)}$  for each  $d \in I'$ . Let  $I_2$  denote the union of subintervals of  $I$  that we obtain if we remove from  $I$  each clean interval  $I' = [a', b']$ . Inductively, we do the same transformation for each maximally large interval  $I'$  of  $I_2$  such that  $q_{\text{OPT}}(d) = \bar{q}^{(2)}$  for each  $d \in I'$ , obtaining a set  $I_3$ . We continue for each  $k' \in \{3, 4, \dots, K\}$ .

If for a value  $d \in [0, \ell(\text{OPT}))$  we did not define a value  $q'(d)$ , then we define  $q'(d) := q_{\text{OPT}}(d)$ . Let  $\text{OPT}'$  denote a tour in which we traverse the edges of  $\text{OPT}$  at the speeds given by  $q'$ . Observe that  $q'(d) \leq q_{\text{OPT}}(d)$  for each  $d \in [0, \ell(\text{OPT}))$ . Using that the speeds are powers of 2 and geometric sum arguments, one can show that  $\text{OPT}' \in O(\text{OPT})$ , see Lemma 13 in the appendix.

We want to argue that  $w(T^*) \in O(\text{OPT}')$ . Consider first the edges  $e \in T^*$  of the form  $e = (u(v, i), u(v', i))$  for two vertices  $v, v'$  and an  $i \in \{1, \dots, K+1\}$ . Let  $T_1^*$  denote the set of all these edges and let  $T_2^* := T^* \setminus T_1^*$ . We first want to show that  $w(T_1^*) \in O(\text{OPT}')$ . Let  $C = \{e_1, e_2, \dots\}$  be a connected component of  $T_1^*$ . Then there exists an  $i \in \{1, \dots, K+1\}$  such that for each edge  $e_j \in C$  there are two vertices  $v_j, v'_j \in V$  such that  $e_j = (u(v_j, i), u(v'_j, i))$ . By definition of  $\text{OPT}'$  and  $w(e_j)$ , we have that  $w(e_j)$  is at most the time that in  $\text{OPT}'$  the head of the convoy needs to move from  $v_j$  to  $v'_j$ . Repeating this argumentation for each edge  $e_j \in T_1^*$ , this implies that  $w(T_1^*) \in O(\text{OPT}')$ .

It remains to argue that  $w(T_2^*) \in O(\text{OPT}')$ . Assume that  $V = \{v_1, \dots, v_n\}$ . For each vertex  $v_j \in V$ , denote by  $d_j$  the distance traveled by the head of the convoy when it reaches  $v_j$  for the first time. Assume w.l.o.g. that the vertices are ordered such that  $d_j < d_{j'}$  for each  $j, j'$  with  $j < j'$ . This partitions the  $[0, \ell(\text{OPT}'))$  into intervals of the form  $[d_j, d_{j+1})$  with  $j \in \{1, \dots, n\}$ , where for convenience we define  $d_{n+1} := \ell(\text{OPT}')$ . Consider an edge  $e \in T_2^*$ . By definition of  $T^*$ , there are vertices  $v_j, v_{j+1}, v_{j+2}$  and speeds  $\bar{q}^{(i)}, \bar{q}^{(i')}$  such that  $e = (u(v_{j+1}, i), u(v_{j+1}, i'))$  and at some point during the interval  $[d_j, d_{j+1})$  or at some point during the interval  $[d_{j+1}, d_{j+2})$  (measured in the total distance that the head of the convoy has traveled) the convoy travels at speed  $\min\{\bar{q}^{(i)}, \bar{q}^{(i')}\}$  due to some edge  $e'$ . We charge  $w(e) = L / \min\{\bar{q}^{(i)}, \bar{q}^{(i')}\}$  to the slow-down of the convoy due to  $e'$ . The remainder of the argumentation follows via geometric sum arguments, addressing the case when we charge two edges of different speeds that are traversed (partially) at the same time by the convoy in  $\text{OPT}'$ .

To this end, for each speed  $\bar{q}^{(i)}$  we define  $\mathcal{I}^{(i)}$  to be the family of maximal intervals  $I$  of interval  $[0, \ell(\text{OPT}'))$  during which the convoy travels at speed  $\bar{q}^{(i)}$  or slower (again, measured in the total distance that the head of the convoy has traveled). For each speed  $\bar{q}^{(i)}$  we define  $|\mathcal{I}^{(i)}|$  to be the total length of the intervals in  $\mathcal{I}^{(i)}$  and we observe that  $|\mathcal{I}^{(i)}| / \bar{q}^{(i)} \leq \text{OPT}'$ . Since the speeds are powers of 2, via a geometric sum argument we also obtain that  $\sum_i |\mathcal{I}^{(i)}| / \bar{q}^{(i)} \leq O(\text{OPT}')$ . We argue now how to charge  $w(T_2^*)$  to  $\sum_i |\mathcal{I}^{(i)}| / \bar{q}^{(i)}$ . Consider an edge  $e = (u(v_{j+1}, i), u(v_{j+1}, i')) \in T_2^*$  as defined above and corresponding speeds  $\bar{q}^{(i)}, \bar{q}^{(i')}$ . Let  $i^*$  be such that  $\bar{q}^{(i^*)} = \min\{\bar{q}^{(i)}, \bar{q}^{(i')}\}$ . Then there is an interval  $I = [a, b) \in \mathcal{I}^{(i^*)}$  such that  $a \in [d_j, d_{j+2})$  or  $b \in [d_j, d_{j+2})$ . We have that  $w(e) \leq (b-a) / \bar{q}^{(i^*)}$ . Thus, we charge  $w(e)$  to  $I$ . In this way, each interval  $I$  in each set  $\mathcal{I}^{(i)}$  is charged at most twice, and we obtain that hence the total charge of all edges in  $T_2^*$  amounts to  $w(T_2^*) = \sum_{e \in T_2^*} w(e) \leq 2 \sum_i |\mathcal{I}^{(i)}| / \bar{q}^{(i)} \leq O(\text{OPT}')$ . Thus,  $w(T^*) = w(T_1^*) + w(T_2^*) \in O(\text{OPT}')$ . ◀

We conclude that the time the convoy needs to traverse our computed tour  $T$  is bounded by  $O(w(T')) = O(K \cdot w(T^*)) = O(\log n \cdot \text{OPT})$ . This yields our  $O(\log n)$ -approximation algorithm for the Convoy-TSP problem.

► **Theorem 2.** *There is a polynomial time  $O(\log n)$ -approximation algorithm for Convoy-TSP.*

## 4 Convoy-Chinese Postman Problem

In this section we discuss Convoy-CPP, which is defined on undirected graphs. First we show that Convoy-CPP is NP-hard, also for directed graphs. Note that CPP can be solved to optimality in polynomial time for both undirected [2] and directed graphs [3]. For mixed graphs, CPP is already NP-hard [9].

► **Theorem 4.** *The Convoy-CPP is NP-hard, both for directed and undirected graphs, even when all edge lengths are equal to 1 and  $L = 2$ .*

**Proof.** To show NP-hardness for undirected graphs we use a reduction from the Hamiltonian path problem [4]. In this problem we are given a graph  $G = (V, E)$  and the question is whether there exists a Hamiltonian path, i.e., a path visiting all vertices exactly once, in  $G$ .

Let  $I$  be an instance of the Hamiltonian path problem. We create graph  $G'$  by copying the graph  $G$  of  $I$ . Each edge in  $G'$  has a length of 1 and a speed of  $M = 4m + 1$ , where  $m$  is the number of edges in  $G$ . We also pick an arbitrary vertex from this graph as the special vertex  $r$ . Now we create a new vertex in  $G'$  for every vertex in  $G$ , and connect it to its corresponding vertex. Each new edge has length 1 and speed 1. Further, we set  $L$  equal to 2. We will show that there is a solution in which the convoy needs less than  $3n + 2$  time units to traverse all edges if and only if the original graph has a Hamiltonian path.

Suppose that  $G$  contains a Hamiltonian path. The convoy can first visit all edges with speed  $M$  spending at most  $2m/M$  time units. It can then move to one of the endpoints of the Hamiltonian path, spending at most  $m/M$  time units. The convoy can now follow the Hamiltonian path, where it visits the edges with speed 1 upon arrival at a vertex. Since this will decrease the speed to 1, the head of the convoy will arrive at the other endpoint of the Hamiltonian path after  $3n - 1$  more time units. While the head of the convoy heads back to  $r$ , the convoy will travel at speed 1 for  $L = 2$  more time units. After that, the tail of the convoy still needs to travel a distance of at most  $m$ , while the convoy is at speed  $M$ . In total, the time spent by the convoy is at most

$$\frac{2m}{M} + \frac{m}{M} + 3n - 1 + L + \frac{m}{M} = \frac{4m}{M} + 3n - 1 + L < 3n + 2.$$

On the other hand, suppose that  $G$  does not contain a Hamiltonian path. In any solution, traversing the edges of speed 1 takes  $2n$  time units. After the last visit to an edge with speed 1, the convoy moves at speed 1 for another  $L = 2$  time units. The convoy also travels at speed 1 for at least one time unit between consecutive visits of speed 1 edges, since none of these edges are adjacent. Moreover, since  $G$  does not contain a Hamiltonian path, there needs to be at least one pair of speed 1 edges that are visited consecutively for which the convoy travels at speed 1 for at least two time units. In total, the time spent in any solution is at least  $2n + n + L = 3n + 2$ . To show NP-hardness for directed graphs we reduce from the directed Hamiltonian path problem [4] and perform a similar reduction. ◀

Next, we will relate the approximability of Convoy-CPP and Convoy-TSP. As a corollary, we also obtain an  $O(\log n)$ -approximation for Convoy-CPP on general graphs.

► **Lemma 12.** *For a given instance of Convoy-CPP whose optimal solution has value  $\alpha$ , we can construct in polynomial time an instance of Convoy-TSP whose optimal solution has value  $\beta$  such that  $\beta \leq \alpha \leq 2\beta$ .*

**Proof.** First, replace each edge  $(u, v) \in E$  with edges  $(u, w)$ ,  $(w, u)$ ,  $(w, v)$ , and  $(v, w)$ , where each of these edges gets length  $\ell((u, v))/2$ , and speed  $q((u, v))$ . The optimal solution to the given instance of Convoy-CPP is also feasible for the constructed instance of Convoy-TSP,

and thus  $\beta \leq \alpha$ . On the other hand, consider an optimal solution for the created Convoy-TSP instance with value  $\beta$ . If the resulting tour turns at  $w$  for some edge  $(u, v)$ , we can adjust this tour to one that visits the original edge  $(u, v)$  fully at once, by moving back and forth between  $u$  and  $v$  when we visit  $w$ . Since this will not slow down the convoy, and the distance traveled along  $(u, v)$  is doubled, the total time spent will increase with at most a factor 2. This implies that  $\alpha \leq 2\beta$ . ◀

Together with our algorithm from Section 3, this yields our  $O(\log n)$ -approximation algorithm for Convoy-CPP.

► **Theorem 3.** *There is a polynomial time  $O(\log n)$ -approximation algorithm for Convoy-CPP.*

We remark that with similar arguments one can obtain an  $O(\log n)$ -approximation algorithm for Convoy-CPP on directed or on mixed graphs.

---

## References

- 1 Pierre Chardaire, Geoff P. McKeown, S.A. Verity-Harrison, and S.B. Richardson. Solving a time-space network formulation for the convoy movement problem. *Operations research*, 53(2):219–230, 2005.
- 2 Jack Edmonds. The Chinese postman problem. *Operations Research*, 13:73–77, 1965.
- 3 Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical programming*, 5(1):88–124, 1973.
- 4 Michael R. Garey and David S. Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.
- 5 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.
- 6 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 32–45, 2021.
- 7 Michael Hart Moore. On the fastest route for convoy-type traffic in flowrate-constrained networks. *Transportation Science*, 10(2):113–124, 1976.
- 8 Dong Hwan Oh, R. Kevin Wood, and Young Hoon Lee. Optimal interdiction of a ground convoy. *Military Operations Research*, 23(2):5–18, 2018.
- 9 Christos H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23:544–554, 1976.
- 10 Marta Pascoal, M. Eugénia V. Captivo, and João C.N. Clímaco. A comprehensive survey on the quickest path problem. *Annals of Operations Research*, 147(1):5–21, 2006.
- 11 P.N. Ram Kumar and T.T. Narendran. A mathematical approach for variable speed convoy movement problem (CMP). *Defense & Security Analysis*, 25(2):137–155, 2009.
- 12 Martin Skutella. An introduction to network flows over time. In William Cook, László Lovász, and Jens Vygen, editors, *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 451–482. Springer, Berlin, Heidelberg, 2009.
- 13 Petr Slavík. The errand scheduling problem, 1997. Technical report.
- 14 Ola Svensson, Jakub Tarnawski, and László A Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *Journal of the ACM (JACM)*, 67(6):1–53, 2020.
- 15 Vera Traub and Jens Vygen. An improved approximation algorithm for the asymmetric traveling salesman problem. *SIAM Journal on Computing*, 51(1):139–173, 2022.

## A

 Details for the proof of Lemma 11

► **Lemma 13.** *We have that  $\text{OPT}' \in O(\text{OPT})$ .*

**Proof.** Let  $I' = [a, b] \subseteq I$  denote a maximally large subinterval of  $I$  such that  $q_{\text{OPT}}(d) = \bar{q}^{(1)}$  for each  $d \in I'$ . We have that  $|I'| \geq L$  since  $\bar{q}^{(1)}$  is the lowest speed. Due to  $I'$  we defined for each  $d \in [a', \min\{a' + L, b\})$  and for each  $d \in [\max\{b' - L, a\}, b')$  that  $q'(d) := \bar{q}^{(1)}$ . We charge the time traveled with speed  $\bar{q}^{(1)}$  on  $[a', \min\{a' + L, b\}) \cup [\max\{b' - L, a\}, b')$  to  $I'$ . For our charging scheme, we say that the clean interval  $[a', b')$  gives  $4L/\bar{q}^{(1)}$  credits to each adjacent maximally large non-clean interval, and we charge this to  $[a', b')$ . We do the same argumentation and charging for each interval that became clean in this iteration.

Suppose by induction that we have given as input a maximally large non-clean interval  $\tilde{I}$  such that, for some  $k$ ,  $q_{\text{OPT}}(d) \geq \bar{q}^{(k)}$  for each  $d \in \tilde{I}$  that received  $4L/\bar{q}^{(k-1)}$  credits from the previous iterations. Let  $I' = [a, b] \subseteq \tilde{I}$  denote a maximally large subinterval of  $\tilde{I}$  such that  $q_{\text{OPT}}(d) = \bar{q}^{(k)}$  for each  $d \in I'$ . Suppose that due to  $I'$  we define for each  $d \in [a', \min\{a' + L, b\})$  and for each  $d \in [\max\{b' - L, a\}, b')$  that  $q'(d) := \bar{q}^{(k)}$ .

First, if  $|I'| \geq L$  then we charge the times traveled with speed  $\bar{q}^{(k)}$  on  $[a', \min\{a' + L, b\}) \cup [\max\{b' - L, a\}, b')$  to  $I'$ . Also, we charge  $I'$  in order to give  $4L/\bar{q}^{(k)}$  credits to each adjacent maximally large non-clean interval that is adjacent to  $[a', b')$ .

Second, if  $|I'| < L$  then  $I'$  must be at the left or the right end of  $\tilde{I}$ . In that case we charge those times to  $2L/\bar{q}^{(k-1)}$  of the  $4L/\bar{q}^{(k-1)}$  credits that we received from the previous iteration. We use the remaining  $2L/\bar{q}^{(k-1)} = 4L/\bar{q}^{(k)}$  credits to give  $4L/\bar{q}^{(k)}$  credits to each maximally large non-clean interval that is adjacent to  $[a', b')$ . Together, this implies the lemma. ◀



# Ordinal Measures of the Set of Finite Multisets

Isa Vialard 

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

---

## Abstract

Well-partial orders, and the ordinal invariants used to measure them, are relevant in set theory, program verification, proof theory and many other areas of computer science and mathematics. In this article we focus on a common data structure in programming, finite multisets of some well partial order. There are two natural orders one can define on the set of finite multisets of a partial order: the multiset embedding and the multiset ordering. Though the maximal order type of these orders is already known, other ordinal invariants remain mostly unknown. Our main contributions are expressions to compute compositionally the width of the multiset embedding and the height of the multiset ordering. Furthermore, we provide a new ordinal invariant useful for characterizing the width of the multiset ordering.

**2012 ACM Subject Classification** Theory of computation → Complexity theory and logic; Theory of computation → Logic and verification; Theory of computation → Proof theory; Theory of computation → Program reasoning

**Keywords and phrases** Well-partial order, finite multisets, termination, program verification

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.87

**Related Version** *Full Version*: <https://arxiv.org/abs/2302.09881>

## Introduction

**Measuring partial orders** is useful in many domains, from set theory to proof theory, including infinitary combinatorics, program verification, rewriting theory, proof automation and many more.

There are intuitive notions of measure for a partial order when it is finite: its cardinal obviously, but also its height (the length of a maximal chain) or its width (the length of a maximal antichain). Similar notions exist for infinite partial orders, as long as they are *well partial orders* (**wpo**), i.e., well-founded partial orders with no infinite antichains [10, 12]. Two such notions are the *ordinal height*, which is the order type of a maximal chain, and the *maximal order type* (**mot**), which is the order type of a maximal linearisation, a notion introduced by De Jongh and Parikh in order to measure hierarchies of functions [6]. These are transfinite measures, hence we call them *ordinal invariants*. Kříž and Thomas introduced alternative characterizations for **mot** and ordinal height, which naturally led to the definition of a third ordinal invariant, *ordinal width* [11]. Less studied than its counterparts, the width of a **wpo** relates to its antichains, even though it cannot be defined as the order type of a maximal antichain. While exploring techniques for program termination, Blass and Gurevich rediscovered these characterizations with a game-theoretical point of view [4].

Ordinal invariants of **wpos** have also been used to prove complexity bounds. In the last decade there has been a flurry of complexity results for the verification of well-structured transition systems (**wsts**), i.e., transition systems whose set of configurations is a **wpo** and whose transitions respect this ordering [5]. When a **wsts** is based on a **wpo**  $X$  of maximal order type  $\omega^\alpha$ , one can expect the complexity of coverability to be in  $H_{\omega^\alpha}$  in the Hardy hierarchy, or in  $F_\alpha$  in the fast-growing hierarchy [9]. This bound can be refined by looking at controlled antichains instead of controlled bad sequences [14], thus bounding complexity with width instead of maximal order type.



© Isa Vialard;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 87; pp. 87:1–87:15

Leibniz International Proceedings in Informatics



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

**Computing ordinal invariants compositionally.** Many wpos underlying wsts are built from classical operations on simpler wpos whose invariants are known. This has spurred new interest in measuring the ordinal invariants of various well-ordered data structures: De Jongh and Parikh computed the *mot* of the disjoint sum and the Cartesian product of wpos [6]. Schmidt then computed the *mot* of word embedding and homeomorphic tree embedding on a wpo [13]. Abraham and Bonnet pursued this line of study by computing the height of Cartesian product, but also the width of disjoint sum and lexicographic product [1]. For a complete survey of these results see [8], where Džamonja et al. computed the ordinal invariants of the lexicographic product, but also the height of the multiset word and tree embeddings.

**Finite multisets.** In this article, we study the ordinal invariants of the set of finite multisets. Multisets, also called “bags”, a common data structure in computer science. Informally, a finite multiset over a set  $X$  is a finite subset of  $X$  where an element can appear finitely many times. For instance,  $\langle a, a, b \rangle$  denote the multiset where  $a$  appears twice and  $b$  once. One can see the set of finite multisets on a wpo as the set of finite words quotiented by the equivalence relation “equality up to some permutation”. It comes down to describing a multiset as a word where the order of terms is irrelevant. A finite multiset can be represented by a function from  $X$  to  $\mathbb{N}$  with finite support, which associates its multiplicity with each element.

Two orderings are classically defined on the finite multisets of any ordered set. The first one is the *multiset ordering*, which often appears in rewriting theory and automation of termination proofs [7]. The other, less-known, ordering is the *multiset embedding*, or term ordering as it is called in [15]. It was presented by Aschenbrenner and Pong as a natural extension of the embedding order over finite words [3].

Some invariants of these two orderings have already been measured: Van der Meeren, Rathjen, and Weiermann [15] built on [17] to compute the *mot* of the set of finite multisets on a wpo  $X$  ordered with the multiset ordering, and provided a new proof for the expression of the *mot* of the multiset embedding computed in [18]. Džamonja et al. [8] proved that the height of the multiset embedding is equal to the height of the set of finite words ordered with word embedding. It is noteworthy that these three results give expressions that are functional in (i.e., can be expressed as a function of) the *mot* and height of  $X$ . However, the height of the multiset ordering still needs to be determined, and the width remains unstudied for both orderings.

**Our contributions.** In this article, we provide functional expressions for the width of the multiset embedding (Theorem 2.1) and the height of the multiset ordering (Theorem 3.1).

We further show that the width of the multiset ordering cannot be expressed as a function of the three ordinal invariants (Example 3.2). Nonetheless, we get around this issue by introducing a fourth ordinal invariant, the *friendly order type* (Definition 3.3), in which the width of the multiset ordering is functional (Theorem 3.4). We then proceed to investigate and compute this new ordinal invariant.

## 1 Definitions and state of the art

### 1.1 Width, height and maximal order type

A sequence  $x_1, \dots, x_n, \dots$  on a partial order  $(X, \leq_X)$  is *good* when there exist  $i < j$  such that  $x_i \leq_X x_j$ , otherwise it is a *bad* sequence. An *antichain* is a sequence whose elements are pairwise incomparable.

A *well partial order* (**wpo**) is a partial order that has no infinite bad sequences. Equivalently, a **wpo** is a partial order that is both well-founded (i.e. no infinite strictly decreasing sequences) and has no infinite antichains.

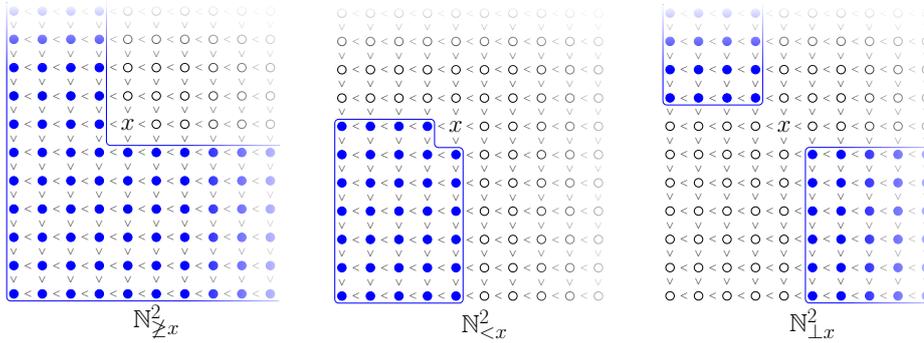
Let  $(X, \leq_X)$  be a **wpo**. We often write just  $X$  when  $\leq_X$  is understood. The trees  $Bad(X)$ ,  $Dec(X)$  and  $Ant(X)$  are defined as the sets of bad sequences, strictly decreasing sequences, and antichains of  $X$ , respectively, ordered by inverse prefix order (a sequence is smaller than its prefixes) ([11, 8]). The finiteness of bad sequences, strictly decreasing sequences and antichains in a **wpo** implies that these trees are well-founded. Therefore, one can define a notion of rank on these trees: a sequence has rank 0 when it cannot be extended; otherwise its rank is the smallest ordinal strictly larger than the ranks of its extensions. The rank of a tree is the rank of the empty sequence (which is the root of the tree).

The *maximal order type* (or **mot**) of  $X$ , denoted by  $\mathbf{o}(X)$ , is defined as the rank of  $Bad(X)$ . Similarly, the *height*  $\mathbf{h}(X)$  and the *width*  $\mathbf{w}(X)$  of  $X$  are defined as the ranks of  $Dec(X)$  and  $Ant(X)$ , respectively. Together,  $\mathbf{o}(X)$ ,  $\mathbf{h}(X)$  and  $\mathbf{w}(X)$  are called the *ordinal invariants* of  $X$ .

For any **wpo**  $X$ ,  $Dec(X)$  and  $Ant(X)$  are subtrees of  $Bad(X)$ . Thus  $\mathbf{h}(X) \leq \mathbf{o}(X)$  and  $\mathbf{w}(X) \leq \mathbf{o}(X)$ .

Let  $x \perp y$  denote that  $x$  and  $y$  are incomparable. For a relation  $*$  among  $\{ \not\leq, <, \perp \}$ , we define the residual  $X_{*x}$  as  $\{ y \in X : y * x \}$ . This definition can be extended to subsets  $S \subseteq X$ :  $X_{*S} \stackrel{\text{def}}{=} \{ y \in X : \forall x \in S, y * x \}$ .

► **Example 1.1.** In Figure 1, you can see the residuals at  $x = (4, 6)$  of  $\mathbb{N} \times \mathbb{N}$  ordered component-wise.



■ **Figure 1** Residuals of  $\mathbb{N}^2$  at  $(4, 6)$ .

Since the rank of the empty sequence is the smallest ordinal strictly larger than the ranks of the sequences of length 1, the definitions of **mot**, height and width can be reformulated inductively through the following *residual equations*:

$$\mathbf{o}(X) = \sup_{x \in X} (\mathbf{o}(X_{\not\leq x}) + 1) \tag{Res-o}$$

$$\mathbf{h}(X) = \sup_{x \in X} (\mathbf{h}(X_{< x}) + 1) \tag{Res-h}$$

$$\mathbf{w}(X) = \sup_{x \in X} (\mathbf{w}(X_{\perp x}) + 1) \tag{Res-w}$$

With these equations we can compute easily the ordinal invariants of  $\mathbb{N}^2$ . For instance, observe that  $(\mathbb{N}^2_{< x})$  is finite for any  $x \in \mathbb{N}^2$ , so its height is finite but can be arbitrarily big. Hence  $\mathbf{h}(\mathbb{N}^2) = \omega$ .

### 1.2 Ordinal arithmetic

We suppose well-known the notions of sum, product, subtraction, natural sum, natural product on ordinals, denoted with  $+$ ,  $\cdot$ ,  $-$ ,  $\oplus$ ,  $\otimes$  [2]. However, let us recall some definitions and notations that might be less familiar to the reader.

An ordinal  $\alpha$  is *indecomposable* iff for any  $\delta, \gamma < \alpha$ , we have  $\delta \oplus \gamma < \alpha$ . Equivalently,  $\alpha$  is indecomposable when there is an ordinal  $\beta$  such that  $\alpha = \omega^\beta$ .  $\alpha$  is an  $\epsilon$ -number when  $\alpha = \omega^\alpha$ .

The *Hessenberg-based product*  $\alpha \odot \beta$  is defined inductively as follows [1]:

$$\alpha \odot 0 = 0, \quad \alpha \odot (\beta + 1) = (\alpha \odot \beta) \oplus \alpha, \quad \alpha \odot \beta = \sup\{ \alpha \odot \gamma : \gamma < \beta \} \text{ for limit } \beta.$$

This definition ensures that  $\alpha \cdot \beta \leq \alpha \odot \beta \leq \alpha \otimes \beta$ .

For any ordinal  $\alpha = \omega^{\alpha_1} + \dots + \omega^{\alpha_n}$ , let  $\hat{\alpha} \stackrel{\text{def}}{=} \omega^{\alpha'_1} + \dots + \omega^{\alpha'_n}$ , where  $\alpha'_i$  is  $\alpha_i + 1$  when  $\alpha_i$  is the sum of an  $\epsilon$ -number and a finite ordinal, otherwise  $\alpha'_i = \alpha_i$ .

For any ordinals  $\alpha, \beta$ , let  $\alpha \hat{\oplus} \beta \stackrel{\text{def}}{=} \sup\{ \alpha' \oplus \beta' : \alpha' < \alpha, \beta' < \beta \}$ .

### 1.3 Ordinal invariants of basic data structures

For any wpos  $P, Q$ , the *disjoint sum*  $P \sqcup Q$  is the disjoint union of  $P$  and  $Q$  ordered such that elements of  $P$  and  $Q$  cannot be compared together, whereas the *direct sum*  $P + Q$  is the disjoint union of  $P$  and  $Q$  ordered such that for all  $p \in P, q \in Q, p \leq q$ . For a family of wpos  $(A_i)_{i < \alpha}$ , let  $\Sigma_{i < \alpha} A_i$  denote the direct sum of the  $A_i$ s along the ordinal  $\alpha$ .

The *Cartesian product*  $P \times Q$  is the set of pairs  $(p, q) \in P \times Q$  where elements are compared component-wise. The *lexicographic product* of  $P$  along  $Q$ , written  $P \cdot Q$ , has the same support as  $P \times Q$ , with a different ordering:  $(p, q) \leq_{P \cdot Q} (p', q')$  iff  $q <_Q q'$ , or  $q = q'$  and  $p \leq_P p'$ .

Sums and products are the most basic operations on wpos one can find. Their ordinal invariants are easy to compute compositionally (see Table 1), with the notable exception of the width of the Cartesian product which cannot be expressed as a function of the ordinal invariants its factors [16].

■ **Table 1** How to compute ordinal invariants compositionally, [8]. See Section 1.2 for definitions of  $\hat{\oplus}$  and  $\odot$ .

Space $X$	M.O.T. $\mathfrak{o}(X)$	Height $\mathfrak{h}(X)$	Width $\mathfrak{w}(X)$
$A \sqcup B$	$\mathfrak{o}(A) \oplus \mathfrak{o}(B)$	$\max(\mathfrak{h}(A), \mathfrak{h}(B))$	$\mathfrak{w}(A) \oplus \mathfrak{w}(B)$
$A + B$	$\mathfrak{o}(A) + \mathfrak{o}(B)$	$\mathfrak{h}(A) + \mathfrak{h}(B)$	$\max(\mathfrak{w}(A), \mathfrak{w}(B))$
$A \times B$	$\mathfrak{o}(A) \otimes \mathfrak{o}(B)$	$\mathfrak{h}(A) \hat{\oplus} \mathfrak{h}(B)$	(Not functional)
$A \cdot B$	$\mathfrak{o}(A) \cdot \mathfrak{o}(B)$	$\mathfrak{h}(A) \cdot \mathfrak{h}(B)$	$\mathfrak{w}(A) \odot \mathfrak{w}(B)$

### 1.4 Comparing wpos

A widely-used and intuitive relation between wpos is the *reflection* relation. A mapping between wpos  $f: (A, \leq_A) \rightarrow (B, \leq_B)$  is a *reflection* if  $f(x) \leq_B f(y)$  implies  $x \leq_A y$ , i.e. it is a morphism from  $(A, \not\leq_A)$  to  $(B, \not\leq_B)$ . Let  $A \leftrightarrow B$  denote that there is a reflection from  $A$  to  $B$ .

However, in this article, we prefer to use the stronger notions of augmentations and substructures.

► **Definition 1.2** (Substructure, augmentation). *A wpo  $(A, \leq_A)$  is a substructure of a wpo  $(B, \leq_B)$  whenever  $A \subseteq B$  and  $\leq_A$  is the restriction of  $\leq_B$  to  $A$ . This relation is written  $A \leq_{st} B$ . Similarly  $(A, \leq_A)$  is an augmentation of  $(B, \leq_B)$  whenever  $A = B$  and  $\leq_B \subseteq \leq_A$ . We write this relation  $A \geq_{aug} B$ .*

Obviously,  $A \leq_{st} B$  or  $A \geq_{aug} B$  imply  $A \leftrightarrow B$ .

We often abuse these notations and write  $A \leq_{st} B$  (resp.  $B \leq_{aug} A$ ) to mean that  $A$  is isomorphic to a substructure (resp. an augmentation) of  $B$ .

We denote by  $A \equiv B$  that  $(A, \leq_A)$  is isomorphic to  $(B, \leq_B)$ .

In this article, when we consider a subset  $Y$  of a wpo  $X$ , it is understood that  $Y \leq_{st} X$ , i.e.  $Y$  is ordered with  $\leq_X$  restricted to the subset.

These notions of augmentations and substructures allow us to compare the ordinal invariants of wpos.

► **Lemma 1.3.** *Let  $A$  and  $B$  be wpos.*

*If  $A \leq_{st} B$  then  $\mathbf{i}(A) \leq \mathbf{i}(B)$  for  $\mathbf{i} \in \{ \mathbf{o}, \mathbf{h}, \mathbf{w} \}$ .*

*If  $A \geq_{aug} B$  then  $\mathbf{o}(A) \leq \mathbf{o}(B)$  and  $\mathbf{w}(A) \leq \mathbf{w}(B)$ . However  $\mathbf{h}(A) \geq \mathbf{h}(B)$ .*

The substructure and augmentation relations are monotonous through most operations on wpos. For instance, if  $A \leq_{st} A'$ , then  $A \times B \leq_{st} A' \times B$ .

An ordinal, as defined by Von Neumann, is the linear wpo that contains all smaller ordinals. Thus augmentations and substructures relations can also be used to compare directly ordinals to wpos. The following result is well-known:

► **Proposition 1.4.** *For any wpo  $X$ ,  $\mathbf{h}(X)$  and  $\mathbf{o}(X)$  are the largest ordinals such that  $\mathbf{h}(X) \leq_{st} X$  and  $\mathbf{o}(X) \geq_{aug} X$ .*

## 1.5 Orderings on the set of finite multisets

We assume familiarity with finite multisets and the associated operations as used in [17]: union, intersection and subtraction, denoted by  $\cup, \cap$  and  $\setminus$ , respectively. Let  $\langle x_1, \dots, x_n \rangle$  denote the finite multiset that contains the elements  $x_1, \dots, x_n$  (they do not have to be distinct). For any  $k \in \mathbb{N}$ ,  $m \times k$  means the union of  $k$  copies of  $m$ . Let  $|m|$  denote the number of elements of a multiset  $m$ .

There are two main orderings classically defined on the set of finite multisets  $M(X)$  of a partial order  $X$ :

► **Definition 1.5** (Multiset embedding [18]). *The multiset embedding on  $M(X)$ , also known as the term ordering, is defined as:*

$m \leq_{\diamond} m'$  iff there exists  $f : m \rightarrow m'$  injective such that for any  $x \in m$ ,  $x \leq f(x)$ .

► **Definition 1.6** (Multiset ordering [17]). *The multiset ordering on  $M(X)$  is defined as:*

$m \leq_r m' \iff m = m' \text{ or } \forall x \in m \setminus (m \cap m'), \exists y \in m' \setminus (m \cap m'), x < y$ .

We write  $M^{\diamond}(X)$  for  $(M(X), \leq_{\diamond})$  and  $M^r(X)$  for  $(M(X), \leq_r)$ .

The multiset ordering and the multiset embedding are both augmentations of the word embedding on  $X^*$  the set of finite words on  $X$ . Therefore, according to Higman's lemma [10],  $M^{\diamond}(X)$  and  $M^r(X)$  are wpos when  $X$  is. Moreover  $M^{\diamond}(X) \leq_{aug} M^r(X)$ , as was observed by Aschenbrenner and Pong [3].

Observe that if  $X$  is a linear ordering, then  $M^r(X)$  is linear, while  $M^\diamond(X)$  is not as long as  $X$  has more than two elements.

► **Proposition 1.7** (Transformation equations). *For any wpos  $A$  and  $B$ ,*

$$M^*(A \sqcup B) \equiv M^*(A) \times M^*(B) \text{ for } * \in \{\diamond, r\}, \quad (\text{Trans-1})$$

$$M^r(A + B) \equiv M^r(A) \cdot M^r(B), \quad (\text{Trans-2})$$

$$M^\diamond(A + B) \leq_{\text{aug}} M^\diamond(A) \cdot M^\diamond(B). \quad (\text{Trans-3})$$

► **Lemma 1.8** (Width of  $M(X)$  on  $\Gamma_k$ ). *For any  $k < \omega$ , we denote by  $\Gamma_k$  the wpo that contains  $k$  incomparable elements. Then  $\mathbf{w}(M^\diamond(\Gamma_k)) = \mathbf{w}(M^r(\Gamma_k)) = \omega^{k-1}$ .*

**Proof.** Since  $M^\diamond(\Gamma_1) \equiv M^r(\Gamma_1) \equiv \omega$ , Equation (Trans-1) tells us that  $M^\diamond(\Gamma_k)$  and  $M^r(\Gamma_k)$  are both isomorphic to the  $k$ -fold Cartesian product  $\omega \times \cdots \times \omega$ . This special case of the width of a Cartesian product is known [16]:  $\mathbf{w}(\omega \times \cdots \times \omega) = \omega^{k-1}$ . ◀

The augmentation and substructure relations are monotone with respect to the multiset ordering and multiset embedding:

► **Proposition 1.9.** *Let  $A, B$  be two wpos. Then  $A \leq_{\text{st}} B$  implies  $M^\diamond(A) \leq_{\text{st}} M^\diamond(B)$  and  $M^r(A) \leq_{\text{st}} M^r(B)$ . Moreover,  $A \geq_{\text{aug}} B$  implies that  $M^\diamond(A) \geq_{\text{aug}} M^\diamond(B)$  and  $M^r(A) \geq_{\text{aug}} M^r(B)$ .*

### Ordinal invariants of the set of finite multisets

Van der Meeren, Rathjen and Weiermann computed the  $\text{mot}$  of  $M^\diamond(X)$  and  $M^r(X)$ .

► **Theorem 1.10** (Mot of multiset embedding [15, 18]). *For any wpo  $X$ ,  $\mathbf{o}(M^\diamond(X)) = \omega^{\widehat{\mathbf{o}(X)}}$ .*

► **Theorem 1.11** (Mot of multiset ordering [15, 17]). *For any wpo  $X$ ,  $\mathbf{o}(M^r(X)) = \omega^{\mathbf{o}(X)}$ .*

Observe that  $\omega^{\mathbf{o}(X)} \leq \omega^{\widehat{\mathbf{o}(X)}}$ , as one would expect since  $M^r(X) \geq_{\text{aug}} M^\diamond(X)$ . Furthermore, we expect that  $\mathbf{w}(M^r(X)) \leq \mathbf{w}(M^\diamond(X))$ , while  $\mathbf{h}(M^r(X)) \geq \mathbf{h}(M^\diamond(X))$ .

► **Theorem 1.12** (Height of the multiset embedding [8]). *Let  $X$  be a wpo. Then  $\mathbf{h}(M^\diamond(X)) = \mathbf{h}^*(X)$ , where*

$$\mathbf{h}^*(X) \stackrel{\text{def}}{=} \begin{cases} \mathbf{h}(X) & \text{if } \mathbf{h}(X) \text{ is infinite and indecomposable,} \\ \mathbf{h}(X) \cdot \omega & \text{otherwise.} \end{cases}$$

## 1.6 A tool to compute the width: Quasi-incomparable subsets

Of all three ordinal invariants, the width is the less studied, since it has been introduced more recently, and also the hardest invariant to study for lack of tools.

A powerful tool to analyse the width of a wpo is the notion of *quasi-incomparable* subsets of a wpo, which was first introduced in [16] for the Cartesian product of several ordinals.

For any subsets  $Y, Z$  of  $X$ , let  $Y \perp Z$  denote that for every  $y \in Y, z \in Z, y \perp z$ .

► **Definition 1.13.** *Let  $A$  be a wpo, and  $A_1, \dots, A_n$  be  $n$  subsets of  $A$ . Then  $(A_i)_{1 \leq i \leq n}$  is a quasi-incomparable family of subsets of  $A$  iff for any  $i < n$ , for any finite  $Y \subseteq A_1 \cup \cdots \cup A_i$ , there exists  $A'_{i+1} \subseteq A_{i+1}$  such that  $A'_{i+1} \perp Y$  and  $A'_{i+1} \equiv A_{i+1}$ .*

This definition is slightly more restrictive than the one in [16], which only required that  $w(A'_{i+1}) = w(A_{i+1})$ .

The idea behind these quasi-incomparable subsets is that sometimes one can slice a wpo  $A$  into simpler subsets  $A_1, \dots, A_n$  whose width is known, such that  $Ant(A_n) + \dots + Ant(A_1)$  is embedded in  $Ant(A)$ . Intuitively, it means that one can combine antichains of  $A_1, \dots, A_n$  into one antichain of  $A$ .

This entails a practical relation between the widths of  $A$  and its subsets:

► **Lemma 1.14** ([16]). *Let  $(A_i)_{i \leq n}$  be a quasi-incomparable family of subsets of  $A$ . Then  $w(A) \geq w(A_n) + \dots + w(A_1)$ .*

## 2 Ordinal width of the multiset embedding

In this section we compute the width of  $M^\diamond(X)$  for any wpo  $X$ , which happens to be functional in the width of  $X$ :

► **Theorem 2.1** (Width of the multiset embedding). *For any wpo  $X$ ,  $w(M^\diamond(X)) = \omega^{\widehat{o(X)}-1}$ . (See Section 1.2 for the definition of  $\widehat{\alpha}$ .)*

It is already known that, in some cases, the width of the multiset embedding reaches its mot.

► **Lemma 2.2** ([8]). *If  $o(X)$  is infinite and indecomposable,  $w(M^\diamond(X)) = o(M^\diamond(X))$ .*

We focus for now on the set of finite multisets on a linear wpo, i.e., an ordinal. Let us treat first the case of successor ordinals.

► **Lemma 2.3.** *For any successor ordinal  $\alpha = \beta + 1$ ,  $w(M^\diamond(\alpha)) \geq w(M^\diamond(\beta)) \cdot \omega$ .*

**Proof.** We denote with  $M_{>k}^\diamond(X)$  the subset  $\{ m \in M^\diamond(X) : |m| > k \}$  for any  $k \in \mathbb{N}$  of  $M^\diamond(X)$  for any wpo  $X$ , for any  $k < \omega$ .

Let  $m_n \stackrel{\text{def}}{=} \langle \beta \rangle \times n$  for any  $n \in \mathbb{N}$ . According to Equation (Res-w),

$$\begin{aligned} w(M^\diamond(\alpha)) &= \sup\{ w(M^\diamond(\alpha)_{\perp m}) + 1 : m \in M^\diamond(\alpha) \} \\ &\geq \sup\{ w(M^\diamond(\alpha)_{\perp m_n}) + 1 : n \in \mathbb{N} \}. \end{aligned}$$

Let  $M_k \stackrel{\text{def}}{=} \{ \langle \beta \rangle \times (n - k) \cup m : m \in M_{>k}^\diamond(\beta) \}$  for  $k \in [1, n]$ . These subsets of  $M^\diamond(\alpha)$  are actually subsets of  $M^\diamond(\alpha)_{\perp m_n}$ : for all  $m \in M_k$ ,  $m \perp m_n$  since  $|m| > |m_n|$ . Observe also that for any  $k \in [1, n]$ ,  $M_k \equiv M^\diamond(\beta)$ .

Moreover,  $(M_k)_{k \in [1, n]}$  is a quasi-incomparable family of subsets of  $M^\diamond(\alpha)_{\perp m_n}$ : for any  $i < n$ , for any finite  $Y \subset M_1 \cup \dots \cup M_i$ , let  $s(Y) = \max\{|m|, m \in Y\}$ . Observe that  $M_{i+1}$  contains  $M_{i+1} \cap M_{>s(Y)}^\diamond(\beta)$  which is incomparable to  $Y$ , and isomorphic to  $M_{i+1}$ .

Therefore,  $w(M^\diamond(\alpha)_{\perp m_n}) \geq w(M_n) + \dots + w(M_1) = w(M^\diamond(\beta)) \cdot n$  according to Lemma 1.14. Thus  $w(M^\diamond(\alpha)) \geq \sup\{ w(M^\diamond(\beta)) \cdot n + 1 : n \in \mathbb{N} \} = w(M^\diamond(\beta)) \cdot \omega$ . ◀

► **Lemma 2.4.** *For any infinite ordinal  $\alpha$ ,  $w(M^\diamond(\alpha)) = o(M^\diamond(\alpha))$ .*

**Proof.** We already know that  $w(M^\diamond(\alpha)) \leq o(M^\diamond(\alpha))$ . We prove the lower bound by induction on  $\alpha$ :

■ If  $\alpha$  is indecomposable, see Lemma 2.2.

- If  $\alpha = \beta + 1$ , then according to Lemma 2.3,

$$\begin{aligned} \mathbf{w}(M^\circ(\alpha)) &\geq \mathbf{w}(M^\circ(\beta)) \cdot \omega \\ &= \mathbf{o}(M^\circ(\beta)) \cdot \omega && \text{by induction hypothesis,} \\ &= \omega^{\widehat{\beta+1}} = \omega^{\widehat{\beta+1}} = \mathbf{o}(M^\circ(\alpha)) && \text{according to Theorem 1.10.} \end{aligned}$$

- If  $\alpha = \beta + \omega^\rho$  with  $\beta, \omega^\rho < \alpha$  and  $\rho > 0$ , then according to the transformation equation Trans-3,  $M^\circ(\alpha) \leq_{\text{aug}} M^\circ(\beta) \cdot M^\circ(\omega^\rho)$ . Hence according to Lemma 1.3 and Table 1,

$$\begin{aligned} \mathbf{w}(M^\circ(\alpha)) &\geq \mathbf{w}(M^\circ(\beta)) \odot \mathbf{w}(M^\circ(\omega^\rho)) \\ &= \mathbf{o}(M^\circ(\beta)) \odot \mathbf{o}(M^\circ(\omega^\rho)) && \text{by induction hypothesis,} \\ &= \omega^{\widehat{\beta}} \odot \omega^{\widehat{\omega^\rho}} = \omega^{\widehat{\alpha}} \\ &= \mathbf{o}(M^\circ(\alpha)) && \text{according to Theorem 1.10.} \quad \blacktriangleleft \end{aligned}$$

We can now prove that Lemma 2.4 generalizes to non-linear wpos.

- **Lemma 2.5.** *If  $\mathbf{o}(X)$  is infinite then  $\mathbf{w}(M^\circ(X)) = \mathbf{o}(M^\circ(X))$ .*

**Proof.** Let  $\alpha = \mathbf{o}(X)$ . Then  $X \leq_{\text{aug}} \alpha$  from Proposition 1.4, hence  $M^\circ(X) \leq_{\text{aug}} M^\circ(\alpha)$  according to Lemma 1.3 and Proposition 1.9. Thus

$$\mathbf{w}(M^\circ(\alpha)) \leq \mathbf{w}(M^\circ(X)) \leq \mathbf{o}(M^\circ(X)).$$

Now  $\mathbf{o}(M^\circ(X)) = \omega^{\widehat{\alpha}} = \mathbf{o}(M^\circ(\alpha))$  according to Theorem 1.10. Now with Lemma 2.4  $\mathbf{w}(M^\circ(\alpha)) = \mathbf{o}(M^\circ(\alpha))$ , hence  $\mathbf{w}(M^\circ(X)) = \mathbf{o}(M^\circ(X))$ .  $\blacktriangleleft$

We can also compute the width of  $M^\circ(X)$  when  $X$  is a finite wpo:

- **Lemma 2.6.** *If  $\mathbf{o}(X)$  is finite, then  $\mathbf{w}(M^\circ(X)) = \omega^{\mathbf{o}(X)-1}$ .*

**Proof.** Let  $k = \mathbf{o}(X)$ . Then  $\Gamma_k \leq_{\text{aug}} X \leq_{\text{aug}} k$ , hence  $\mathbf{w}(M^\circ(\Gamma_k)) \geq \mathbf{w}(M^\circ(X)) \geq \mathbf{w}(M^\circ(k))$  thanks to Lemma 1.3. According to Lemma 1.8,  $\mathbf{w}(M^\circ(\Gamma_k)) = \omega^{k-1}$ , and according to Lemma 2.3 applied  $(k-1)$  times,  $\mathbf{w}(M^\circ(k)) \geq \mathbf{w}(M^\circ(1)) \cdot \omega^{k-1} = \omega^{k-1}$ . Therefore  $\mathbf{w}(M^\circ(X)) = \omega^{k-1} = \omega^{\mathbf{o}(X)-1}$ .  $\blacktriangleleft$

This section's main result follows directly from Lemmas 2.5 and 2.6.

**Proof of Theorem 2.1.** If  $\mathbf{o}(X)$  is finite, then  $\widehat{\mathbf{o}(X)} - 1 = \mathbf{o}(X) - 1$ . On the other hand, if  $\mathbf{o}(X)$  is infinite, then  $\widehat{\mathbf{o}(X)} - 1 = \widehat{\mathbf{o}(X)}$ .  $\blacktriangleleft$

### 3 Ordinal height and width of the multiset ordering

For the height of  $M^r(X)$ , we obtain a result similar to Theorem 1.11.

- **Theorem 3.1** (Height of the multiset ordering). *Let  $X$  be a wpo.*

*Then  $\mathbf{h}(M^r(X)) = \omega^{\mathbf{h}(X)}$ .*

**Proof.** Observe that the multiset ordering of any linear ordering is also linear. Thus, for any ordinal  $\alpha$ ,  $M^r(\alpha)$  is isomorphic to  $\omega^\alpha$  (the function  $\langle x_1, \dots, x_n \rangle \mapsto \omega^{x_1} \oplus \dots \oplus \omega^{x_n}$  is an isomorphism).

According to Proposition 1.4,  $X \geq_{\text{st}} \mathbf{h}(X)$ , and thus  $M^r(X) \geq_{\text{st}} M^r(\mathbf{h}(X)) \equiv \omega^{\mathbf{h}(X)}$  (Proposition 1.9). Therefore  $\mathbf{h}(M^r(X)) \geq \omega^{\mathbf{h}(X)}$  according to Lemma 1.3. See the proof of the upper bound in Appendix A.  $\blacktriangleleft$

The width of the multiset ordering is harder to compute, as  $\mathbf{w}(M^r(X))$  is not functional in the ordinal invariants of  $X$ . The following example exhibits two wpos  $X_1$  and  $X_2$ , with identical ordinal invariants, such that  $\mathbf{w}(M^r(X_1)) \neq \mathbf{w}(M^r(X_2))$ .

► **Example 3.2.** Let  $H \stackrel{\text{def}}{=} \Sigma_{n < \omega} \Gamma_n$ . An interesting property of  $H$  is that  $\mathbf{w}(H) = \mathbf{h}(H) = \mathbf{o}(H) = \omega$ . Since  $M^r(H) \geq_{\text{st}} M^r(\Gamma_n)$ , then  $\omega^{n-1} \leq \mathbf{w}(M^r(H)) \leq \mathbf{o}(M^r(H)) = \omega^\omega$  for all  $n < \omega$  according to Lemma 1.8 and Theorem 1.11. Hence  $\mathbf{w}(M^r(H)) = \omega^\omega$ .

Consider  $X_1 = H + H$  and  $X_2 = H + \omega$ , two wpos with the same ordinal invariants:  $\mathbf{o}(X_i) = \mathbf{h}(X_i) = \omega \cdot 2$  and  $\mathbf{w}(X_i) = \omega$  for  $i \in \{1, 2\}$ . According to Equation (Trans-2) and Table 1,  $\mathbf{w}(M^r(X_1)) = \mathbf{w}(M^r(H)) \odot \mathbf{w}(M^r(H)) = \omega^\omega \odot \omega^\omega = \omega^{\omega \cdot 2}$  and  $\mathbf{w}(M^r(X_2)) = \mathbf{w}(M^r(H)) \odot \mathbf{w}(M^r(\omega)) = \omega^\omega \odot 1 = \omega^\omega$ .

Fortunately, we uncovered a new ordinal invariant, defined similarly to the usual invariants, in which the width of the multiset ordering is functional.

► **Definition 3.3** (Friendly order type). *A bad sequence is open-ended if it is empty or of the form  $sx$  where  $s$  is an open-ended sequence and  $x$  has a “friend”<sup>1</sup> in the residual  $X_{\not\geq s}$ , i.e., an element incomparable to  $x$ . For any wpo  $X$ , let  $\text{Bad}_\perp(X)$  be the subtree of  $\text{Bad}(X)$  which contains all open-ended bad sequences. As  $\text{Bad}_\perp(X)$  is a substructure of  $\text{Bad}(X)$ , it has a rank that we denote by  $\mathbf{o}_\perp(X)$  the friendly order type of  $X$  (or fot).*

*This definition can be expressed as the following residual equation:*

$$\mathbf{o}_\perp(X) = \sup_{x \in X, X_{\perp x} \neq \emptyset} (\mathbf{o}_\perp(X_{\not\geq x}) + 1) \quad (\text{Res-f})$$

► **Theorem 3.4.** *For any wpo  $X$ ,  $\mathbf{w}(M^r(X)) = \omega^{\mathbf{o}_\perp(X)}$*

**Proof.** See Appendix B. The proof of Theorem 3.4 is quite technical, and relies on the notion of quasi-incomparable subsets. ◀

## 4 Computing the friendly order type

Like the usual ordinal invariants, the fot can be computed compositionally for some basic operations on wpos:

► **Proposition 4.1.** *For any non empty wpo  $A, B$ ,*

1.  $\mathbf{o}_\perp(A + B) = \mathbf{o}_\perp(A) + \mathbf{o}_\perp(B)$ ,
2.  $\mathbf{o}_\perp(A \sqcup B) = 1 + (\mathbf{o}(A) - 1) \oplus (\mathbf{o}(B) - 1)$ ,

**Proof.**

1. For any sequences  $s_A, s_B$  in  $\text{Bad}_\perp(A), \text{Bad}_\perp(B)$ , the concatenation  $s_B s_A$  is a sequence of  $\text{Bad}_\perp(A + B)$ . Furthermore, any sequence of  $\text{Bad}_\perp(A + B)$  is of this form.
2. For any two sequences  $s_1, s_2$ , let  $s_1 \sqcup s_2$  denote the set of sequences obtained through shuffling  $s_1, s_2$  together (e.g.  $abcad \in aba \sqcup cd$ ). Let  $x_A, x_B$  be two minimal elements of  $A$  and  $B$ . For any sequences  $s_A, s_B$  in  $\text{Bad}(A \setminus \{x_A\}), \text{Bad}(B \setminus \{x_B\})$ , for any  $s \in s_A \sqcup s_B$ , we know that  $s$  and  $s x_A$  and  $s x_B$  are in  $\text{Bad}_\perp(A \sqcup B)$ . Reciprocally, from any  $s \in \text{Bad}_\perp(A \sqcup B)$ , there is a partition  $s_A \in \text{Bad}(A), s_B \in \text{Bad}(B)$  such that  $s \in s_A \sqcup s_B$ . Furthermore, the natural sum of the ranks of  $s_A$  in  $\text{Bad}(A)$  and  $s_B$  in  $\text{Bad}(B)$  is strictly positive.

<sup>1</sup> Can one be friend with one’s superior or inferior? No. Your true friends are those you cannot (and do not have to) compare yourselves with.

## 87:10 Ordinal Measures of the Set of Finite Multisets

Suppose for contradiction sake that  $s_A$  and  $s_B$  have rank 0 in  $Bad(A)$  and  $Bad(B)$ . Let  $s = s'x$ . Then  $(A \sqcup B)_{\succeq s} = \emptyset$  and in particular  $x$  has no friend in  $(A \sqcup B)_{\succeq s'}$ . Thus  $s \notin Bad_{\perp}(A \sqcup B)$ , contradiction.  $\blacktriangleleft$

Observe how friendly order type behaves similarly to  $\text{mot}$ . It is not unusual to have  $\text{fot}$  coincides with  $\text{mot}$ , for instance  $\mathbf{o}_{\perp}(\omega \sqcup \omega) = \mathbf{o}(\omega \sqcup \omega)$  (Proposition 4.1).

To bring this new ordinal invariant closer to familiar grounds, we bound the  $\text{fot}$  of a wpo  $X$  with the  $\text{mot}$  of a special subset of  $X$ , the *stripped* subset.

► **Definition 4.2** (Stripped subset). *The stripped subset of a wpo  $X$ , denoted by  $\text{str}(X)$ , is  $X$  without its friendless elements:*

$$\text{str}(X) \stackrel{\text{def}}{=} \{ x \in X : X_{\perp x} \neq \emptyset \}.$$

Since  $Bad_{\perp}(X)$  is a subtree of  $Bad(\text{str}(X))$ , we know that  $\mathbf{o}_{\perp}(X) \leq \mathbf{o}(\text{str}(X))$ . Here is an example where this inequality is strict:

► **Example 4.3.** Let  $X = \omega \sqcup \{\clubsuit\}$ . Here  $\text{str}(X) = X$ , so  $\mathbf{o}(\text{str}(X)) = \omega + 1$ . However, in  $Bad_{\perp}(X)$ , the singleton  $\clubsuit$  has rank 0, and the singleton  $n$  for any  $n \in \omega$  has rank  $n$ . Therefore  $\mathbf{o}_{\perp}(X) = \omega < \mathbf{o}(\text{str}(X))$ .

Let us show that  $\mathbf{o}(\text{str}(X))$  also appears in a lower bound on  $\mathbf{o}_{\perp}(X)$ , by introducing an alternative characterisation of  $\text{fot}$  as the  $\text{mot}$  of a specific subset.

A *maximal linearisation* is a monotonic function from a wpo  $X$  onto  $\mathbf{o}(X)$ .

► **Definition 4.4** (Friendly subset). *A subset  $X'$  of  $X$  is friendly if there exist a maximal linearisation  $\ell : X' \rightarrow \mathbf{o}(X')$  such that for any bad sequence  $s = x_1, \dots, x_n$  in  $X'$  verifying  $\ell(x_1) > \dots > \ell(x_n)$ ,  $s$  is open-ended. We say that  $\ell$  witnesses the friendly condition.*

Observe that every friendly subset of  $X$  is a substructure of  $\text{str}(X)$ .

For any ordinal  $\alpha$ , let

$$\delta(\alpha) \stackrel{\text{def}}{=} \begin{cases} \alpha & \text{if } \alpha \text{ is limit,} \\ \gamma + \lfloor n/2 \rfloor & \text{if } \alpha = \gamma + n \text{ with } \gamma \text{ limit and } n < \omega. \end{cases}$$

► **Theorem 4.5** (Alternative characterisation of  $\mathbf{o}_{\perp}(X)$ ). *Let  $X$  be a wpo. There exists a friendly subset  $X'$  of  $X$  which maximizes  $\mathbf{o}(X')$ , and  $\mathbf{o}_{\perp}(X) = \mathbf{o}(X')$ . Furthermore,  $\delta(\mathbf{o}(\text{str}(X))) \leq \mathbf{o}_{\perp}(X) \leq \mathbf{o}(\text{str}(X))$ .*

**Proof.** See proof in Appendix C.  $\blacktriangleleft$

► **Example 4.6** (Following on Example 3.2). Remember that  $H \stackrel{\text{def}}{=} \sum_{n < \omega} \Gamma_n$ . Thus  $\text{str}(H) = \sum_{2 \leq n < \omega} \Gamma_n$ , and  $\mathbf{o}(\text{str}(H)) = \mathbf{o}(H) = \omega$ . Consider  $X_1 = H + H$  and  $X_2 = H + \omega$ . Observe that  $\text{str}(X_1) = \text{str}(H) + \text{str}(H)$  whereas  $\text{str}(X_2) = \text{str}(H)$ . Therefore, according to Theorem 4.5,  $\mathbf{o}_{\perp}(X_1) = \omega \cdot 2$  and  $\mathbf{o}_{\perp}(X_2) = \omega$ .

► **Corollary 4.7.** *For any wpo  $X$ , if  $\mathbf{o}(X)$  is limit and  $\mathbf{o}(\text{str}(X)) = \mathbf{o}(X)$ , then  $\mathbf{o}_{\perp}(X) = \mathbf{o}(X)$ .*

The conditions in Corollary 4.7 are often satisfied:

► **Proposition 4.8.** *For any wpo non-empty  $X$ ,  $\mathbf{o}_{\perp}(M^{\circ}(X)) = \mathbf{o}(M^{\circ}(X))$ .*

**Proof.** Observe that  $M^{\circ}(X) = M^{\circ}(X) \setminus \{\emptyset\}$ . Thus  $\mathbf{o}(\text{str}(M^{\circ}(X))) = \mathbf{o}(M^{\circ}(X)) - 1 = \mathbf{o}(M^{\circ}(X))$  (Theorem 1.10). We conclude with Corollary 4.7.  $\blacktriangleleft$

## Conclusion

Table 2 sums up this article’s contributions (in the gray cases) amidst the former state of the art.

■ **Table 2** Ordinal invariants of the set of finite multisets.

Invariants	Multiset embedding of $X$	Multiset ordering of $X$
Mot $\mathbf{o}$	$\omega^{\widehat{\mathbf{o}(X)}}$	$\omega^{\mathbf{o}(X)}$
Height $\mathbf{h}$	$\mathbf{h}^*(X)$	$\omega^{\mathbf{h}(X)}$
Width $\mathbf{w}$	$\omega^{\widehat{\mathbf{o}(X)}-1}$	$\omega^{\mathbf{o}_\perp(X)}$

These results are part of a more general research program (see [8, 16]) aimed at measuring more precisely and more effectively the complexity of wpos used in well-structured systems, termination proofs, and other algorithmic applications.

Investigating the friendly order type is a subject for further research: How does it relate to other concepts? Can it be computed compositionally for more operations? Can we define a class of wpos where friendly order type always coincides with mot?

---

## References

- 1 U. Abraham and R. Bonnet. Hausdorff’s theorem for posets that satisfy the finite antichain property. *Fund. Math.*, 159(1):51–69, 1999. doi:10.4064/fm-159-1-51-69.
- 2 H. J. Altman. Intermediate arithmetic operations on ordinal numbers. *Mathematical Logic Quarterly*, 63(3–4):228–242, 2017. doi:10.1002/malq.201600006.
- 3 M. Aschenbrenner and Wai Yan Pong. Orderings of monomial ideals. *Fundamenta Mathematicae*, 181(1):27–74, 2004.
- 4 A. Blass and Y. Gurevich. Program termination and well partial orderings. *ACM Trans. Computational Logic*, 9(3):1–26, 2008. doi:10.1145/1352582.1352586.
- 5 R. Bonnet, A. Finkel, S. Haddad, and F. Rosa-Velardo. Ordinal theory for expressiveness of well-structured transition systems. *Information and Computation*, 224:1–22, 2013. doi:10.1016/j.ic.2012.11.003.
- 6 D. H. J. de Jongh and R. Parikh. Well-partial orderings and hierarchies. *Indag. Math.*, 39(3):195–207, 1977. doi:10.1016/1385-7258(77)90067-1.
- 7 N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979. doi:10.1145/359138.359142.
- 8 M. Džamonja, S. Schmitz, and Ph. Schnoebelen. On ordinal invariants in well quasi orders and finite antichain orders. In P. Schuster, M. Seisenberger, and A. Weiermann, editors, *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, chapter 2, pages 29–54. Springer, Berlin/Heidelberg, Germany, 2020. doi:10.1007/978-3-030-30229-0\_2.
- 9 D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson’s lemma. In *Proc. 26th IEEE Symp. Logic in Computer Science (LICS 2011), Toronto, Canada, June 2011*, pages 269–278. IEEE Comp. Soc. Press, 2011. doi:10.1109/LICS.2011.39.
- 10 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952. doi:10.1112/plms/s3-2.1.326.
- 11 I. Kříž and R. Thomas. Ordinal types in Ramsey theory and well-partial-ordering theory. In J. Nešetřil and V. Rödl, editors, *Mathematics of Ramsey Theory*, volume 5 of *Algorithms and Combinatorics*, pages 57–95. Springer, Berlin/Heidelberg, Germany, 1990. doi:10.1007/978-3-642-72905-8\_7.

- 12 J. B. Kruskal. Well-quasi-ordering, the Tree Theorem, and Vazsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95(2):210–225, 1960. doi:10.2307/1993287.
- 13 D. Schmidt. Well-partial orderings and their maximal order types. In P. Schuster, M. Seisenberger, and A. Weiermann, editors, *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, chapter 12, pages 351–391. Springer, 2020. doi:10.1007/978-3-030-30229-0\_13.
- 14 S. Schmitz. The parametric complexity of lossy counter machines. In *Proc. 46th Int. Coll. Automata, Languages, and Programming (ICALP 2019), Patras, Greece, July 2019*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 129:1–129:15, Dagstuhl, Germany, 2019. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.129.
- 15 J. Van der Meeren, M. Rathjen, and A. Weiermann. Well-partial-orderings and the big Veblen number. *Archive for Mathematical Logic*, 54(1–2):193–230, 2015. doi:10.1007/s00153-014-0408-5.
- 16 I. Vialard. On the cartesian product of well-orderings. arXiv:2202.07487 [cs.LO], February 2022. Submitted for publication to Order. arXiv:2202.07487.
- 17 A. Weiermann. Proving termination for term rewriting systems. In *Proc. 5th Workshop on Computer Science Logic (CSL '91), Berne, Switzerland, Oct. 1991*, volume 626 of *Lecture Notes in Computer Science*, pages 419–428. Springer, 1991. doi:10.1007/BFb0023786.
- 18 A. Weiermann. A computation of the maximal order type of the term ordering on finite multisets. In *Proc. 5th Conf. Computability in Europe (CiE 2009), Heidelberg, Germany, July 2009*, volume 5635 of *Lecture Notes in Computer Science*, pages 488–498. Springer, 2009. doi:10.1007/978-3-642-03073-4\_50.

### A Proof of Theorem 3.1

We write  $m \overset{\curvearrowright}{\leq}_r m'$  (resp.  $m \overset{\curvearrowright}{<} m'$ ,  $m \overset{\curvearrowright}{\perp} m'$ ) when  $m \cap m' \neq \emptyset$  and  $m \leq_r m'$  (resp  $m < m'$ ,  $m \perp m'$ ). With these new notations, the multiset ordering can be reformulated as follows

► **Definition A.1** (Multiset ordering (reformulated)).  $M^r(X) = (M(X), \leq_r)$  is ordered with the multiset ordering:  $m \leq_r m'$  iff there exists  $m_1, m'_1, m_2$  such that  $m = m_1 \cup m_2$ ,  $m' = m'_1 \cup m_2$ , and  $m_1 \overset{\curvearrowright}{<} m'_1$ .

► **Lemma A.2.** Let  $A = \cup_{i \leq n} A_i$  a set partitioned in  $n$  subsets, for some  $n \in \mathbb{N}$ . Let  $\leq_A$  a well-partial ordering on  $A$ , and  $\leq_{A_i}$  the same ordering restricted to the subset  $A_i$  for  $i \leq n$ . Then

$$\mathbf{h}(A, \leq_A) \leq \bigoplus_{i \leq n} \mathbf{h}(A_i, \leq_{A_i}).$$

**Proof.** From any decreasing sequence  $s$  on  $A$ , one can extract a decreasing sequence  $s_i$  by restricting  $s$  to  $A_i$  for any  $i \leq n$ . By induction on the rank of  $s$  in  $\text{Dec}(A)$ , one shows that  $\text{rk}(s) \leq \bigoplus_{i \leq n} \text{rk}(s_i)$ . ◀

**Proof of Theorem 3.1.** We prove the upper bound by induction on  $\mathbf{h}(X)$ .

If  $\mathbf{h}(X) = 0$  then  $X = \emptyset$  and  $\mathbf{h}(M^r(\emptyset)) = 1 = \omega^0$ .

Suppose that  $X$  is not empty. For any non-empty multiset  $m \in M^r(X)$ , the residual  $M^r(X)_{<_m}$  can be partitioned as follows:

$$M^r(X)_{<_m} = \bigcup_{m_1 + m_2 = m, m_1 \neq \emptyset} \{ m' + m_2 : m' \overset{\curvearrowright}{<} m_1 \}.$$

Note that this union is a partition of the support of  $M^r(X)_{<_m}$ , it does not say anything on the order between the elements of the subsets in the union.

For any non-empty multiset  $m$ , we define  $S_m \stackrel{\text{def}}{=} (\cap_{x \in m} X_{\not\leq x}) \cap (\cup_{x \in m} X_{< x})$  a subset of  $X$ . Thus for any multiset  $m'$  in  $M^r(X)$ ,  $m' \not\leq m$  iff  $m' \in M^r(S_m)$ . Therefore:

$$M^r(X)_{< m} = \bigcup_{m_1+m_2=m, m_1 \neq \emptyset} \{ m' + m_2 : m' \in M^r(S_{m_1}) \}.$$

Observe that  $\mathbf{h}(S_{m_1}) < \mathbf{h}(X)$  by definition of  $S_{m_1}$ . Hence by induction hypothesis  $\mathbf{h}(M^r(S_{m_1})) \leq \omega^{\mathbf{h}(S_{m_1})} < \omega^{\mathbf{h}(X)}$ . Moreover,  $\omega^{\mathbf{h}(X)}$  is indecomposable. Hence according to Lemma A.2:

$$\mathbf{h}(M^r(X)_{< m}) \leq \bigoplus_{m_1+m_2=m, m_1 \neq \emptyset} \mathbf{h}(M^r(\cup_{x \in m_1} X_{< x})) < \omega^{\mathbf{h}(X)}.$$

Therefore  $\mathbf{h}(M^r(X)) \leq \omega^{\mathbf{h}(X)}$  according to Equation (Res-h). ◀

## B Proof of Theorem 3.4

First we prove intermediary lower and upper bounds on the width of the multiset ordering.

► **Lemma B.1.** *Let  $X$  be a wpo. Then*

$$\mathbf{w}(M^r(X)) \geq \sup_{x \in X, n < \omega} \mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \cdot n + 1$$

**Proof.** This proof follows the same structure as the proof of Lemma 2.3: We study the residual of  $M^r(X)$  which contains every element incomparable to some multiset of the form  $\langle x \rangle \times n$ , and slice this residual into a family of quasi-incomparable subsets.

According to Equation (Res-w),

$$\begin{aligned} \mathbf{w}(M^r(X)) &= \sup_{m \in M^r(X)} \mathbf{w}(M^r(X)_{\perp m}) + 1 \\ &\geq \sup_{x \in X, n < \omega} \mathbf{w}(M^r(X)_{\perp \langle x \rangle \times n}) + 1. \end{aligned}$$

For all  $k \in [1, n]$ , let  $M_k = \{ \langle x \rangle \times (n - k) \cup m : m \in M^r(X)_{\perp \langle x \rangle} \}$ . Observe that  $M_k \equiv M^r(X)_{\perp \langle x \rangle}$  for any  $k \in [1, n]$ , and for all  $m \in M_k$ ,  $m \perp \langle x \rangle \times n$ . We claim that  $(M_k)_{k \in [1, n]}$  is a quasi-incomparable family of subsets of  $M^r(X)_{\perp \langle x \rangle \times n}$ : Let  $i < n$  and  $Y$  a finite subset of  $M_1 \cup \dots \cup M_i$ . We define  $m_Y$  and  $M'_{i+1}$  as

$$\begin{aligned} m_Y &\stackrel{\text{def}}{=} \bigcup_{j \leq i} \bigcup_{m \in (M_j \cap Y)} (m \setminus (\langle x \rangle \times (n - j))), \\ M'_{i+1} &\stackrel{\text{def}}{=} \{ \langle x \rangle \times (n - i - 1) \cup m_Y \cup m : m \in M^r(X)_{\perp \langle x \rangle} \}. \end{aligned}$$

Observe that  $M'_{i+1}$  is an isomorphic subset of  $M_{i+1}$ , and  $Y \perp M'_{i+1}$ .

Therefore according to Lemma 1.14,  $\mathbf{w}(M^r(X)_{\perp \langle x \rangle \times n}) \geq \mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \cdot n$ . ◀

► **Lemma B.2.** *Let  $X$  be a wpo. Then*

$$\mathbf{w}(M^r(X)) \leq \sup_{x \in X, n < \omega} \mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \otimes n + 1$$

## 87:14 Ordinal Measures of the Set of Finite Multisets

**Proof.** By definition, for any multisets  $m, m' \in M^r(X)$ ,  $m \perp m'$  means that  $m \neq m'$  and there exists  $m_1, m'_1, m_2$  such that  $m = m_1 \cup m_2$ ,  $m' = m'_1 \cup m_2$  and  $m_1 \overset{\uparrow}{\perp} m'_1$ .

Therefore, the residual  $M^r(X)_{\perp m}$  can be partitioned as an augmentation of a disjoint union:

$$M^r(X)_{\perp m} \geq_{\text{aug}} \bigsqcup_{m_1+m_2=m, m_1 \neq \emptyset} \{ m'_1 + m_2 : m' \in M^r(X), m'_1 \overset{\uparrow}{\perp} m_1 \},$$

which can be reformulated into

$$M^r(X)_{\perp m} \geq_{\text{aug}} \bigsqcup_{m_1 \subseteq m, m_1 \neq \emptyset} M^r(X)_{\overset{\uparrow}{\perp} m_1}$$

where  $M^r(X)_{\overset{\uparrow}{\perp} m_1}$  is the residual  $\{ m' \in M^r(X) : m' \overset{\uparrow}{\perp} m_1 \}$ .

Let us observe this residual:  $m' \overset{\uparrow}{\perp} m_1$  means that  $m'$  and  $m_1$  are disjoint and there exists  $x \in m_1$  such that for all  $y' \in m'$ ,  $x \not\leq y'$ , and there exists  $x' \in m'$  such that for all  $y \in m_1$ ,  $x' \not\leq y$ . In particular  $x' \not\leq x$ . Hence  $m' \overset{\uparrow}{\perp} m_1$  implies there exists  $x \in m_1$  such that  $\langle x \rangle \overset{\uparrow}{\perp} m'$ , which is equivalent to  $\langle x \rangle \perp m'$ . Therefore the support of  $M^r(X)_{\overset{\uparrow}{\perp} m_1}$  is included in a union on  $x \in m_1$  of residuals  $M^r(X)_{\perp \langle x \rangle}$ . With an augmentation we get a disjoint union:

$$M^r(X)_{\overset{\uparrow}{\perp} m_1} \leq_{\text{st} \geq_{\text{aug}}} \bigsqcup_{x \in m_1} M^r(X)_{\perp \langle x \rangle}.$$

Hence according to Table 1,  $M^r(X)_{\perp m} \leq \bigoplus_{m_1 \subseteq m, m_1 \neq \emptyset} \bigoplus_{x \in m_1} \mathbf{w}(M^r(X)_{\perp \langle x \rangle})$ .

Let  $x \in m$  such that  $\mathbf{w}(M^r(X)_{\perp \langle x \rangle})$  is maximal. Then  $\mathbf{w}(M^r(X)_{\perp m}) \leq \mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \otimes n$  for some  $n < \omega$ . Hence according to Equation (Res-w),

$$\mathbf{w}(M^r(X)) = \sup_{m \in M^r(X)} \mathbf{w}(M^r(X)_{\perp m}) + 1 \leq \sup_{x \in X, n < \omega} \mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \otimes n + 1. \quad \blacktriangleleft$$

The bounds provided in Lemmas B.1 and B.2 actually match. Furthermore, they can be reformulated in such a way that the residual on  $M^r(X)$  boils down to a residual on  $X$ :

► **Lemma B.3.** For any non-linear wpo  $X$ ,

$$\mathbf{w}(M^r(X)) = \sup \{ \mathbf{w}(M^r(X_{\not\leq x})) \cdot \omega : x \in X, X_{\perp x} \neq \emptyset \}. \quad (\text{W})$$

**Proof.** For any ordinal  $\alpha$ ,  $\sup_{n < \omega} (\alpha \cdot n + 1) = \sup_{n < \omega} (\alpha \otimes n + 1) = \alpha \cdot \omega$ . Hence according to Lemmas B.1 and B.2,  $\mathbf{w}(M^r(X)) = \sup_{x \in X} (\mathbf{w}(M^r(X)_{\perp \langle x \rangle}) \cdot \omega)$ .

Let  $x \in X$ . If  $X_{\perp x} = \emptyset$ , then  $M^r(X)_{\perp \langle x \rangle} = \emptyset$ . Otherwise let  $y \in X_{\perp x}$ . Observe that, for any  $m \in M^r(X_{\not\leq x})$ ,  $m \cup \langle y \rangle \perp \langle x \rangle$ . Hence

$$\{ \langle y \rangle \cup m : m \in M^r(X_{\not\leq x}) \} \leq_{\text{st}} M^r(X)_{\perp \langle x \rangle} \leq_{\text{st}} M^r(X_{\not\leq x}).$$

Therefore  $\mathbf{w}(M^r(X)_{\perp \langle x \rangle}) = \mathbf{w}(M^r(X_{\not\leq x}))$  if  $X_{\perp x} \neq \emptyset$ , otherwise  $\mathbf{w}(M^r(X)_{\perp \langle x \rangle}) = 0$ . ◀

**Proof of Theorem 3.4.** If  $X$  is linear,  $\text{Bad}_{\perp}(X)$  only contains the empty sequence, hence  $\mathbf{o}_{\perp}(X) = 0$  and  $\mathbf{w}(\text{Mulr}(X)) = 1$ . Otherwise, observe that Equation (W) is quite similar to Equation (Res-f) in its structure. Thus  $\mathbf{w}(M^r(X)) = \omega^{\mathbf{o}_{\perp}(X)}$  follows directly from Equation (W). ◀

## C

 Proof of Theorem 4.5

► **Lemma C.1.** *For any wpo  $X$ , for any maximal linearisation  $\ell : \text{str}(X) \rightarrow \mathbf{o}(\text{str}(X))$ , there exists a friendly subset  $X'$  such that  $\ell$  restricted to  $X'$  verifies the friendly condition, and  $\mathbf{o}(X') \geq \delta(\mathbf{o}(\text{str}(X)))$ .*

**Proof.** We claim that for any  $\beta \leq \mathbf{o}(\text{str}(X))$ , there exists  $X_\beta \subseteq \ell^{-1}(\{\gamma : \gamma < \beta\})$  a friendly subset of  $X$  where  $\ell$  restricted to  $X_\beta$  verifies the friendly condition, such that  $\mathbf{o}(X_\beta) \geq \delta(\beta)$ . In this proof, when we say that a subset is friendly, it is always implied that  $\ell$  restricted to this subset witnesses the friendly condition.

We build the subsets  $(X_\beta)_{\beta \leq \mathbf{o}(\text{str}(X))}$  as follows:

- $X_0 = \emptyset$ ,
- For  $\gamma$  limit,  $X_\gamma = \bigcup_{\beta < \gamma} X_\beta$ ,
- For any  $\beta$ ,  $X_{\beta+1} = X_\beta \cup \ell^{-1}(\beta)$  if friendly, otherwise  $X_{\beta+1} = X_\beta$ .

First observe that  $X_\beta$  is friendly for any  $\beta \leq \mathbf{o}(\text{str}(X))$ . Indeed,  $X_0$  is friendly, and since for any  $\beta < \beta'$ ,  $X_\beta \subseteq X_{\beta'}$ , then the union  $\bigcup_{\beta < \gamma} X_\beta$  for  $\gamma$  limit is friendly by induction.

Let us prove the claim  $\mathbf{o}(X_\beta) \geq \delta(\beta)$ , by showing that for any  $\beta + 2 \leq \mathbf{o}(\text{str}(X))$ , we have  $\mathbf{o}(X_{\beta+2}) > \mathbf{o}(X_\beta)$ . Let  $x = \ell^{-1}(\beta')$  and  $x' = \ell^{-1}(\beta' + 1)$ . Assume for the sake of contradiction that  $X_{\beta+2} = X_\beta$ . This means that neither  $X_\beta \cup \{x\}$  nor  $X_\beta \cup \{x'\}$  are friendly. Hence there exists  $y, y' \in X_\beta$  such that for any  $z \in X$ , we have  $z \perp y \implies z \geq x$  and  $z \perp y' \implies z \geq x'$ . Now because of  $\ell$  we know that  $x \not\geq x'$  and  $y, y' \not\geq x, x'$ . Since  $y, y' \in \text{str}(X)$ , then  $X_{\perp y}$  and  $X_{\perp y'}$  are both non-empty, so actually  $x \perp y$  and  $x' \perp y'$ . And since  $x \not\geq x'$ , we know  $y' < x$ . Therefore  $x \perp x'$ , hence  $y < x'$ . Which leads to a contradiction on the relationship between  $y$  and  $y'$ . ◀

For any friendly subset  $X'$ ,  $\mathbf{o}(X') \leq \mathbf{o}(\text{str}(X))$ , and there exist  $X'$  such that  $\mathbf{o}(X') \geq \delta(\mathbf{o}(\text{str}(X)))$ . Therefore there exists a friendly subset  $X'$  which maximizes  $\mathbf{o}(X')$ .

**Proof of Theorem 4.5.** We say that a bad sequence  $x_1, \dots, x_n$  respects a maximal linearisation  $\ell$  when  $\ell(x_1) > \dots > \ell(x_n)$ . Let  $X'$  be a friendly subset of  $X$  and  $\ell$  a maximal linearisation of  $X'$  that verifies the friendly condition. Observe that  $\text{Bad}(X')$  restricted to sequences that respect  $\ell$  has for rank  $\mathbf{o}(X')$ , and is embedded in  $\text{Bad}_\perp(X)$ . Hence  $\mathbf{o}_\perp(X) \geq \mathbf{o}(X')$ .

We prove the upper bound by induction on  $\mathbf{o}_\perp(X)$ . If  $\mathbf{o}_\perp(X) = 0$  then the only friendly subset of  $X$  is the empty set. Now suppose that  $\mathbf{o}_\perp(X) > 0$ . For any  $x \in \text{str}(X)$ , by induction hypothesis on  $X_{\not\geq x}$ , there exists a friendly subset  $X'$  of  $X_{\not\geq x}$ , with a maximal linearisation  $\ell$  which verifies the friendly condition, such that  $\mathbf{o}(X') \geq \mathbf{o}_\perp(X_{\not\geq x})$ . We extend  $\ell$  to the subset  $X' \cup \{x\}$  of  $X$ , such that  $\ell(x) = \mathbf{o}(X')$ . Now  $\ell$  is a maximal linearisation of  $X' \cup \{x\}$  which verifies the friendly condition, therefore  $\mathbf{o}(X' \cup \{x\})$  is a friendly subset of  $X$  and  $\mathbf{o}(X' \cup \{x\}) > \mathbf{o}_\perp(X_{\not\geq x})$ . Let  $X'$  be a friendly subset of  $X$  which maximizes  $\mathbf{o}(X')$ . Then for any  $x \in \text{str}(X)$ ,  $\mathbf{o}_\perp(X_{\not\geq x}) < \mathbf{o}(X')$ . Therefore  $\mathbf{o}_\perp(X) \leq \mathbf{o}(X')$  according to Equation (Res-f). ◀



# Checking Presence Reachability Properties on Parameterized Shared-Memory Systems

Nicolas Waldburger  

Univ Rennes, Inria, CNRS, IRISA, France

---

## Abstract

We consider the verification of distributed systems composed of an arbitrary number of asynchronous processes. Processes are identical finite-state machines that communicate by reading from and writing to a shared memory. Beyond the standard model with finitely many registers, we tackle round-based shared-memory systems with fresh registers at each round. In the latter model, both the number of processes and the number of registers are unbounded, making verification particularly challenging. The properties studied are generic presence reachability objectives, which subsume classical questions such as safety or synchronization by expressing the presence or absence of processes in some states. In the more general round-based setting, we establish that the parameterized verification of presence reachability properties is PSPACE-complete. Moreover, for the roundless model with finitely many registers, we prove that the complexity drops down to NP-complete and we provide several natural restrictions that make the problem solvable in polynomial time.

**2012 ACM Subject Classification** Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

**Keywords and phrases** Verification, Parameterized models, Distributed algorithms

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.88

**Related Version** *Full Version*: <https://arxiv.org/abs/2306.17476>

**Acknowledgements** Many thanks to Nathalie Bertrand, Nicolas Markey and Ocan Sankur for their invaluable advice.

## 1 Introduction

**Parameterized verification.** Distributed systems consist of multiple processes running in parallel. Verification of such systems is a major topic of modern verification, because of how common these systems are and how difficult their verification has proven to be. Indeed, when multiple processes run asynchronously, the number of relevant interleavings to consider quickly becomes large. An intuitive approach for their verification is to fix the number of processes involved and try to apply classical verification techniques. Another approach is that of parameterized verification, where one aims to prove the more general statement that the property of interest holds for any number of participants. The interest of this approach is threefold. First, it allows to prove that the system is correct regardless of the number of processes. Second, the efficiency of parameterized techniques does not depend on the number of participants, which makes them more suitable for large systems for which classical techniques scale poorly. Third, parameterized verification often yields decidability or better computational complexity for problems that are hard to solve with classical techniques; see for example [14] for a problem that becomes decidable in the parameterized case. In their seminal work [13], German and Sistla consider systems consisting of a leader and arbitrarily many contributors, all of which are finite-state machines communicating via *rendez-vous*. In this setting, the safety verification problem is EXPSpace-complete and the complexity drops down to polynomial time when the leader is removed. Since then, many similar models have been studied, with variations on the expressiveness of the processes and the means of communication in order to capture the large variety of existing distributed algorithms [10, 7].



© Nicolas Waldburger;

licensed under Creative Commons License CC-BY 4.0

48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023).

Editors: Jérôme Leroux, Sylvain Lombardy, and David Peleg; Article No. 88; pp. 88:1–88:15

Leibniz International Proceedings in Informatics



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

**Contributions.** We study parameterized verification of systems where all processes are identical and anonymous finite-state machines that communicate via reading from and writing to a shared memory. The read and write actions are performed non-atomically, meaning that no process may perform a read-write combination while preventing all other processes from acting. Our registers are *initialized* with a special symbol; this assumption is common in parameterized verification of shared-memory systems [8, 1], since some algorithms require initialized registers, *e.g.* [2]. First, we study a model with finitely many registers. This model is inspired by [11] where registers were uninitialized and the verification is restricted to safety properties. In contrast, we study the more general *presence reachability problems*, in which one asks whether one may reach a configuration that satisfies a property. This property takes the form of a Boolean combination of constraints expressing whether there is at least one process in a given state of the finite-state machine. We prove that this problem is NP-complete and we provide several natural restrictions on the process description and on the property that make the problem solvable in polynomial time. We then work on the more general setting of *round-based* shared-memory systems [6], which are designed to model round-based shared-memory algorithms present in the literature, see *e.g.* [2, 15]. In this model, the processes proceed in asynchronous rounds, each round having its own fresh set of registers. The source of infinity is twofold, as the number of processes and the number of registers are both unbounded, making round-based systems particularly challenging to verify. The safety problem was proved to be PSPACE-complete in round-based shared-memory systems [6]. In this article, we go beyond safety by considering a round-based, richer version of the presence reachability problem where the property may quantify existentially and universally over the rounds. Nonetheless, we establish that the round-based presence reachability problem is PSPACE-complete.

**Related work.** Similar models and problems have been studied in the literature. In the shared-memory model (without rounds and without register initialization), the safety problem has been studied extensively with variations on the expressiveness given to the leader and the contributors [11]; in particular, when processes are finite-state machines, the safety problem is shown to be coNP-complete and to decrease to PTIME when the leader is removed. However, this result does not hold when registers are initialized or when the property is more general than safety. A model that has perhaps been more studied is that of *reconfigurable broadcast networks* (RBN), where processes communicate via broadcasting messages that can be received by any of the other processes. This model has similarities with shared-memory systems, although broadcast tends to be simpler (messages disappear after being sent, while written values remain in the registers). A source of inspiration for the first part of our article is the study of reachability problems in RBN [9], where it is shown that the cardinality reachability problem, where one wants to reach a configuration that satisfies cardinality constraints, is PSPACE-complete. When the constraints cannot count processes, this problem is analogous to our presence reachability problem; for RBN, it is shown to be NP-complete, a complexity that we also obtain in our setting. Finally, this complexity drops down to PTIME in RBN when considering the special case of safety. This tractability result no longer holds in the shared-memory world unless we make further assumptions about the number of registers or their initialization. The cube reachability problem is a generalization of the cardinality constraint problem where the initial configuration is also subject to cardinality constraints; this problem is PSPACE-complete both in RBN and in (roundless) asynchronous shared-memory systems [9, 5, 4], although it is unknown whether this remains true when allowing the  $\text{Pre}^*$  and  $\text{Post}^*$  operators in the description of the cubes [4, 3]. While it is

interesting to compare results on RBN with our results on shared-memory systems without rounds, such a comparison is not possible with the more expressive model of round-based shared-memory systems, in particular because the unboundedness in the number of registers has no equivalent in broadcast networks.

Due to space constraints, some details are omitted. These details can be found in the full version of this paper.

## 2 Roundless Register Protocols

In this section, we introduce *register protocols*, a model inspired by [11]. We call these systems *roundless* to distinguish them from *round-based* systems introduced later in this article.

### 2.1 Definitions

► **Definition 1** (Roundless register protocols). *A roundless register protocol is a tuple  $\mathcal{P} = \langle Q, Q_0, \text{dim}, D, \mathbf{d}_0, \Delta \rangle$  where*

- $Q$  is a finite set of states with a distinguished subset of initial states  $Q_0 \subseteq Q$ ;
- $\text{dim} \in \mathbb{N}$  is the number of shared registers;
- $D$  is a finite data alphabet containing the initial symbol  $\mathbf{d}_0$ ;
- $\Delta \subseteq Q \times \mathcal{A} \times Q$  is the set of transitions, where  $\mathcal{A} := \{\text{read}_\alpha(\mathbf{d}) \mid \alpha \in [1, \text{dim}], \mathbf{d} \in D\} \cup \{\text{write}_\alpha(\mathbf{d}) \mid \alpha \in [1, \text{dim}], \mathbf{d} \in D \setminus \{\mathbf{d}_0\}\}$  is the set of actions.

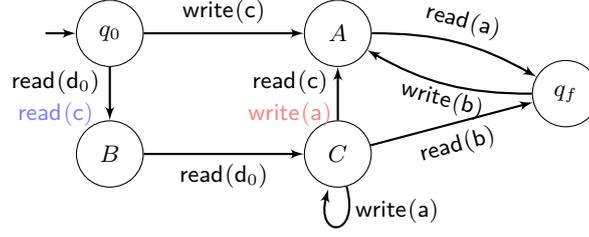
Roundless register protocols are executed on multiple processes that behave asynchronously and can only communicate via reading from and writing to the shared registers. The behavior of a process is described by a finite-state machine. The possible actions of the transitions are reading a symbol from and writing a symbol to one of the  $\text{dim}$  shared registers;  $\mathbf{d} \in D$  denotes the symbol and  $\alpha$  indicates the register on which the action is performed. Each register stores one *symbol* from the finite set  $D$  at a time. Read-write combinations are performed non-atomically, i.e., no process can perform a read-write combination while excluding all other processes. The *size* of the protocol  $\mathcal{P}$  is defined as  $|\mathcal{P}| := |Q| + |D| + |\Delta| + \text{dim}$ . For all  $\alpha \in [1, \text{dim}]$ , we write  $\text{rg}[\alpha]$  for the register of index  $\alpha$ . We also write  $\text{Reg}$  for the set  $\{\text{rg}[\alpha] \mid \alpha \in [1, \text{dim}]\}$  of all registers.

Processes are assumed to have no identifiers so they are identical anonymous agents. Therefore, a *configuration* is a pair  $\gamma = \langle \mu, \vec{d} \rangle \in \mathbb{N}^Q \times D^{\text{Reg}}$  such that  $0 < \sum_{q \in Q} \mu(q) < \infty$ . Let  $\text{st}(\gamma) := \mu$  which indicates the number of processes in each state, and  $\text{data}(\gamma) := \vec{d}$  mapping to each register its symbol: for all  $r \in \text{Reg}$ ,  $\text{data}(\gamma)(r)$  is the symbol contained in register  $r$  in  $\gamma$ . Let  $\Gamma := \mathbb{N}^Q \times D^{\text{Reg}}$  denote the set of all configurations. Let  $\text{supp}(\gamma) := \{q \in Q \mid \text{st}(\gamma)(q) > 0\}$  denote the support of the multiset  $\text{st}(\gamma)$ . We write  $\oplus$  and  $\ominus$  the operations on multisets that add and remove elements, respectively. A configuration is *initial* if all processes are in states from  $Q_0$  while all registers have value  $\mathbf{d}_0$ . We denote by  $\text{Init}_c$  the set of initial configurations (the letter  $c$  stands for “concrete” as opposed to “abstract” configurations defined later). Formally,  $\text{Init}_c := \{\gamma \mid \text{st}(\gamma) \subseteq Q_0, \text{data}(\gamma) = \mathbf{d}_0^{\text{Reg}}\}$ .

Given  $\gamma, \gamma' \in \Gamma$ ,  $\gamma'$  is a *successor* of  $\gamma$  when there exists  $\delta = (q, a, q') \in \Delta$  such that  $\text{st}(\gamma)(q) > 0$ ,  $\text{st}(\gamma') = (\text{st}(\gamma) \ominus \{q\}) \oplus \{q'\}$  and:

- if  $a = \text{read}_\alpha(\mathbf{d})$  then  $\text{data}(\gamma)(\text{rg}[\alpha]) = \mathbf{d}$  and  $\text{data}(\gamma') = \text{data}(\gamma)$ ,
- if  $a = \text{write}_\alpha(\mathbf{d})$  then  $\text{data}(\gamma')(\text{rg}[\alpha]) = \mathbf{d}$  and  $\forall \alpha' \neq \alpha, \text{data}(\gamma')(\text{rg}[\alpha']) = \text{data}(\gamma)(\text{rg}[\alpha'])$ .

In that case, we write  $\gamma \xrightarrow{\delta} \gamma'$  or simply  $\gamma \rightarrow \gamma'$ , which is called a *step*. A *concrete execution* is a sequence  $\pi = \gamma_0, \delta_1, \gamma_1, \dots, \gamma_{l-1}, \delta_l, \gamma_l$  such that for all  $i$ ,  $\gamma_i \xrightarrow{\delta_{i+1}} \gamma_{i+1}$ . We write  $\gamma_0 \xrightarrow{*} \gamma_l$



■ **Figure 1** An example of a protocol.

for the existence of such an execution.  $\gamma'$  is *reachable from*  $\gamma$  when  $\gamma \xrightarrow{*} \gamma'$ . Given a set  $C$  of configurations, we write  $\text{Reach}_c(C) := \{\gamma' \mid \exists \gamma \in C, \gamma \xrightarrow{*} \gamma'\}$ . A configuration is *reachable* when it is in  $\text{Reach}_c(\text{Init}_c)$ .

► **Example 2.** Figure 1 provides an example of a roundless register protocol  $\mathcal{P}$  with  $D = \{d_0, a, b, c\}$ ,  $Q_0 = \{q_0\}$  and  $\dim = 1$ , hence read and write actions are implicitly on register  $\alpha = 1$ . The red and blue labels are to be ignored for now.

The set of initial configurations is  $\text{Init}_c := \{\langle q_0^n, d_0 \rangle \mid n \geq 1\}$ . The following execution with two processes witnesses that  $\langle q_f \oplus C, a \rangle \in \text{Reach}_c(\text{Init}_c)$ :

$$\begin{aligned} \langle q_0^2, d_0 \rangle &\xrightarrow{(q_0, \text{read}(d_0), B)} \langle q_0 \oplus B, d_0 \rangle \xrightarrow{(B, \text{read}(d_0), C)} \langle q_0 \oplus C, d_0 \rangle \xrightarrow{(q_0, \text{write}(c), A)} \\ \langle A \oplus C, c \rangle &\xrightarrow{(C, \text{write}(a), C)} \langle A \oplus C, a \rangle \xrightarrow{(A, \text{read}(a), q_f)} \langle q_f \oplus C, a \rangle. \end{aligned} \quad \lrcorner$$

## 2.2 Reachability Problems

Our first problem of interest is the *coverability problem* (COVER):

COVER FOR ROUNDLESS REGISTER PROTOCOLS

**Input:** A roundless register protocol  $\mathcal{P}$ ,  $q_f \in Q$

**Question:** Does there exist  $\gamma \in \text{Reach}_c(\text{Init}_c)$  such that  $\text{st}(\gamma)(q_f) > 0$ ?

Note that, because the model is parameterized, a witness execution of COVER may have an arbitrarily large number of processes. The dual is the *safety problem*, the answer to which is yes when an error state cannot be covered regardless of the number of processes. A similar problem is the *target problem* (TARGET) where processes must synchronize at  $q_f$ :

TARGET FOR ROUNDLESS REGISTER PROTOCOLS

**Input:** A roundless register protocol  $\mathcal{P}$ ,  $q_f \in Q$

**Question:** Does there exist  $\gamma \in \text{Reach}_c(\text{Init}_c)$  s.t. for all  $q \neq q_f$ ,  $\text{st}(\gamma)(q) = 0$ ?

► **Remark 3.** TARGET is harder than COVER: consider the reduction in which one adds a loop on  $q_f$  writing a joker symbol which, from any state, may be read to reach  $q_f$ .

*Presence constraints* are Boolean combinations (with  $\wedge$ ,  $\vee$  and  $\neg$ ) of atomic propositions of the form “ $q$  populated” with  $q \in Q$ , or of the form “ $r$  contains  $d$ ” with  $r \in \text{Reg}$  and  $d \in D$ . A presence constraint is interpreted over a configuration  $\gamma$  by interpreting “ $q$  populated” as true if and only if  $\text{st}(\gamma)(q) > 0$  and “ $r$  contains  $d$ ” as true if and only if  $\text{data}(\gamma)(r) = d$ . Note that presence constraints cannot refer to how many processes are on a given state. We write  $\gamma \models \phi$  when configuration  $\gamma$  satisfies presence constraint  $\phi$ .

► **Example 4.** If  $Q = \{q_1, q_2, q_3\}$ ,  $\dim = 2$ ,  $D = \{d_0, a, b\}$  and  $\phi := (q_1 \text{ populated}) \vee ((q_2 \text{ populated}) \wedge (\text{rg}[1] \text{ contains } a))$  then  $\langle q_1 \oplus q_3, d_0^2 \rangle \models \phi$ ,  $\langle q_2^2, (a, b) \rangle \models \phi$  but  $\langle q_2^2, b^2 \rangle \not\models \phi$ .  $\lrcorner$

The *Presence Reachability Problem* (PRP) generalizes both COVER and TARGET. It corresponds to the cardinality reachability problem for cardinality constraints restricted to  $CC[\geq 1, = 0]$  studied for broadcast protocols [9].

PRP FOR ROUNDLESS REGISTER PROTOCOLS

**Input:** A roundless register protocol  $\mathcal{P}$ , a presence constraint  $\phi$

**Question:** Does there exist  $\gamma \in \text{Reach}_c(\text{Init}_c)$  such that  $\gamma \models \phi$ ?

The formula  $\phi$  automatically makes PRP NP-hard, since one can encode the SAT problem. Therefore, we also consider the *DNF Presence Reachability Problem* (DNF-PRP), in which  $\phi$  is in *disjunctive normal form*. COVER and TARGET are special cases of DNF-PRP, with  $\phi = (q_f \text{ populated})$  for COVER and  $\phi = \bigwedge_{q \neq q_f} \neg(q \text{ populated})$  for TARGET.

► **Example 5.** Consider again the protocol  $\mathcal{P}$  defined in Figure 1.  $(\mathcal{P}, q_f)$  is a positive instance of COVER, as proved in Example 2. Let  $\mathcal{P}_{\text{blue}}$  be the protocol obtained from  $\mathcal{P}$  by changing to  $\text{read}(c)$  the label of the transition from  $q_0$  to  $B$  (in blue in Figure 1).  $(\mathcal{P}_{\text{blue}}, q_f)$  is a negative instance of COVER. In fact, a process can only get to  $B$  if  $c$  has been written to the register, and then  $d_0$  can no longer be read so no process may go to state  $C$ ,  $a$  cannot be written and no process may go from  $A$  to  $q_f$ .

$(\mathcal{P}, q_f)$  is a negative instance of TARGET: to leave  $A$ , one needs to read  $a$ , hence must have a process on state  $C$ , and to leave  $C$ , one must read  $b$  which would force us to send a process to  $A$ . Let  $\mathcal{P}_{\text{red}}$  be the protocol obtained from  $\mathcal{P}$  by changing to  $\text{write}(a)$  the label of the transition from  $C$  to  $A$  (in red in Figure 1).  $(\mathcal{P}_{\text{red}}, q_f)$  is a positive instance of TARGET:

$$\langle q_0^2, d_0 \rangle \xrightarrow{(q_0, \text{read}(d_0), B)} \langle q_0 \oplus B, d_0 \rangle \xrightarrow{(B, \text{read}(d_0), C)} \langle q_0 \oplus C, d_0 \rangle \xrightarrow{(q_0, \text{write}(c), A)} \langle A \oplus C, c \rangle \xrightarrow{(C, \text{write}(a), A)} \langle A^2, a \rangle \xrightarrow{(A, \text{read}(a), q_f)} \langle A \oplus q_f, a \rangle \xrightarrow{(A, \text{read}(a), q_f)} \langle q_f^2, a \rangle.$$

Let  $\phi := \neg(C \text{ populated}) \wedge ((\text{rg contains } a) \vee [(\text{rg contains } b) \wedge \neg(A \text{ populated})])$ .  $\phi$  is a presence constraint and  $(\mathcal{P}, \phi)$  is a negative instance of PRP. Indeed, if  $a$  is in the register, then  $C$  must be populated and if  $b$  is in the register, then  $A$  must be populated. ◻

## 2.3 Abstract Semantics

In this subsection, we define an abstraction of the semantics that is sound and complete with respect to PRP. The intuition of this abstraction is that the exact number of processes in a given state is not relevant. Indeed, register protocols, thanks to non-atomicity, enjoy a classical monotonicity property named copycat property.

► **Lemma 6** (Copycat). *Consider  $\gamma_1, \gamma_2, q_2$  such that  $\gamma_1 \xrightarrow{*} \gamma_2, q_2 \in \text{supp}(\gamma_2)$ . There exists  $q_1 \in \text{supp}(\gamma_1)$  s.t.  $\langle \text{st}(\gamma_1) \oplus q_1, \text{data}(\gamma_1) \rangle \xrightarrow{*} \langle \text{st}(\gamma_2) \oplus q_2, \text{data}(\gamma_2) \rangle$ .*

An *abstract configuration* is a pair  $\sigma = \langle \text{st}(\sigma), \text{data}(\sigma) \rangle \in 2^Q \times D^{\text{Reg}}$  such that  $\text{st}(\sigma) \neq \emptyset$ . The set of *initial configurations* is  $\text{Init}_a := \{\langle S, d_0^{\dim} \rangle \mid S \subseteq Q_0\}$ . Given a concrete configuration  $\gamma$ , the projection  $\text{abst}(\gamma)$  is the abstract configuration  $\langle \text{supp}(\gamma), \text{data}(\gamma) \rangle$ . Let  $\Sigma := 2^Q \times D^{\text{Reg}}$  denote the set of abstract configurations. For  $\sigma, \sigma' \in \Sigma$ ,  $\sigma'$  is the *successor* of  $\sigma$  when there exists  $\delta = (q, a, q') \in \Delta$  such that  $q \in \text{st}(\sigma)$ , either  $\text{st}(\sigma') = \text{st}(\sigma) \cup \{q'\}$  or  $\text{st}(\sigma') = (\text{st}(\sigma) \setminus \{q\}) \cup \{q'\}$ , and: if  $a = \text{read}_\alpha(d)$  then  $\text{data}(\sigma)(\text{rg}[\alpha]) = d$  and  $\text{data}(\sigma) = \text{data}(\sigma')$ , and if  $a = \text{write}_\alpha(d)$  then  $\text{data}(\sigma')(\text{rg}[\alpha]) = d$  and for all  $\alpha' \neq \alpha$ ,  $\text{data}(\sigma')(\text{rg}[\alpha']) = \text{data}(\sigma)(\text{rg}[\alpha'])$ . Again, we denote such a step by  $\sigma \xrightarrow{\delta} \sigma'$  or  $\sigma \rightarrow \sigma'$ . Note that one could equivalently define  $\sigma \xrightarrow{\delta} \sigma'$  by:  $\sigma \xrightarrow{\delta} \sigma' \iff \exists \gamma, \gamma' \in \Gamma, \gamma \xrightarrow{\delta} \gamma'$  and  $\text{abst}(\gamma) = \sigma, \text{abst}(\gamma') = \sigma'$ . This notion of abstraction is classical in parameterized verification of systems with identical anonymous agents that enjoy monotonicity properties. Note, however, that this semantics is

non-deterministic: one could have  $\sigma'' \neq \sigma'$  such that  $\sigma \xrightarrow{\delta} \sigma'$  and  $\sigma \xrightarrow{\delta} \sigma''$ . This alternative corresponds to whether all processes in  $q$  take the transition ( $\text{st}(\gamma') = (\text{st}(\gamma) \setminus \{q\}) \cup \{q'\}$ ) or only some ( $\text{st}(\gamma') = \text{st}(\gamma) \cup \{q'\}$ ). We define *abstract executions* similarly to concrete ones, and denote them using  $\rho$ . We also define the *reachability set*  $\text{Reach}_a(A)$  and the notion of *coverability* as in the concrete case. This abstraction is sound and complete for PRP:

► **Proposition 7** (Soundness and completeness of the abstraction). *For all  $S \subseteq Q$ ,  $\vec{d} \in D^{\text{Reg}}$ :*

$$(\exists \gamma \in \text{Reach}_c(\text{Init}_c) : \text{supp}(\gamma)=S, \text{data}(\gamma)=\vec{d}) \iff (\exists \sigma \in \text{Reach}_a(\text{Init}_a) : \text{st}(\sigma)=S, \text{data}(\sigma)=\vec{d}).$$

The intuition of the proof is the following: any concrete configuration can easily be lifted into an abstract one. Conversely, any abstract execution may be simulated in the concrete semantics for a sufficiently large number of processes by using the copycat property.

Given a presence constraint  $\phi$  and  $\sigma \in \Sigma$ , we define whether  $\sigma$  satisfies  $\phi$ , written  $\sigma \models \phi$ , in a natural way. Given a concrete configuration  $\gamma$ , one has  $\gamma \models \phi$  if and only if  $\text{abst}(\gamma) \models \phi$ . Indeed,  $\gamma$  and  $\text{abst}(\gamma)$  have the same populated states and register values. Therefore, there exists  $\gamma \in \text{Reach}_c(\text{Init}_c)$  such that  $\gamma \models \phi$  if and only if there exists  $\sigma \in \text{Reach}_a(\text{Init}_a)$  such that  $\sigma \models \phi$ : one can consider PRP directly in the abstract semantics.

### 3 Complexity Results for Roundless Register Protocols

In this section, we provide complexity results for the presence reachability problems defined above in the general case and in some restricted cases. Throughout the rest of the section, all configurations and executions are implicitly abstract.

#### 3.1 NP-Completeness of the General Case

First, all problems defined in the previous section are NP-complete.

► **Proposition 8.** *COVER, TARGET, DNF-PRP and PRP for roundless register protocols are all NP-complete.*

**Proof.** First, we prove that all four problems are in NP. It suffices to prove it for PRP, as the three other problems reduce to it.

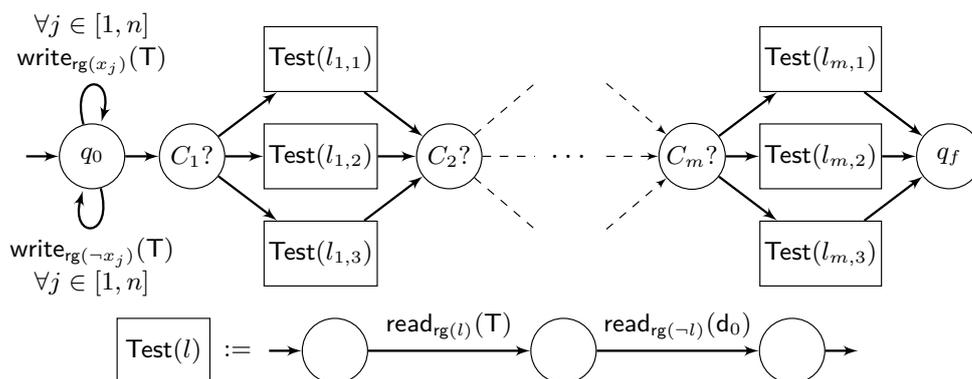
Let  $\rho : \sigma_0 \xrightarrow{*} \sigma$  an abstract execution, we simply prove the existence of  $\rho' : \sigma_0 \xrightarrow{*} \sigma$  of length at most  $4|Q|$ . To obtain  $\rho'$  from  $\rho$ , we iteratively:

- remove any read step that is non-deserting and does not cover a new location,
- remove any write step that is non-deserting, does not populate a new state and whose written symbol is never read,
- make non-deserting any deserting step whose source state is populated again later in  $\rho$ .

In  $\rho'$ , at most  $|Q|$  steps populate a new state and at most  $|Q|$  steps are deserting. This implies that there are at most  $2|Q|$  read steps, therefore, at most  $2|Q|$  write steps whose written value is actually read. In total, this bounds the number of steps by  $4|Q|$ . In particular, for PRP, we can look for an execution of length less than  $4|Q|$  which can be guessed in polynomial time.

We prove NP-hardness of COVER, as it reduces to the three other problems.

The proof is by a reduction from 3-SAT. Consider a 3-CNF formula  $\phi = \bigwedge_{i=1}^m l_{i,1} \vee l_{i,2} \vee l_{i,3}$  over  $n$  variables  $x_1, \dots, x_n$  where, for all  $i \in [1, m]$ , for all  $k \in [1, 3]$ ,  $l_{i,k} \in \{x_j, \neg x_j \mid j \in [1, n]\}$ . We define a roundless register protocol  $\mathcal{P}_{\text{SAT}}(\phi)$  with a distinguished state  $q_f$  which is coverable if and only if  $\phi$  is satisfiable. In  $\mathcal{P}_{\text{SAT}}(\phi)$ , one has  $D = \{d_0, \top\}$  and  $\dim = 2n$ , there are two registers for each variable  $x_i$ ,  $\text{rg}(x_i)$  and  $\text{rg}(\neg x_i)$ . The protocol is represented on Figure 2.



■ **Figure 2** The protocol  $\mathcal{P}_{\text{SAT}}(\phi)$  for NP-hardness of COVER.

While any register may be set to  $\top$  thanks to the loops on  $q_0$ , a register set to  $\top$  can never be set back to  $d_0$ .  $l$  is considered true if  $\text{rg}(l)$  is set to  $\top$  while  $\text{rg}(\neg l)$  still has value  $d_0$ . Suppose that the instance of 3-SAT is positive, *i.e.*,  $\phi$  is satisfiable by some assignment  $\nu$ . Consider an execution that writes  $\top$  exactly to all  $\text{rg}(l)$  with  $l$  true in  $\nu$ . For each clause, one of the three literals is true in  $\nu$ . Therefore the execution may cover  $C_i?$  for all  $i$  so it may cover  $q_f$  and the instance of COVER is positive. Conversely, if the instance COVER is positive, there exists an execution  $\rho : \sigma_0 \xrightarrow{*} \sigma$  with  $\sigma_0 \in \text{Init}_a$  and  $q_f \in \text{st}(\sigma)$ . Consider  $\nu$  that assigns to each variable  $x$  value true if  $\text{rg}(x)$  is written before  $\text{rg}(\neg x)$  in  $\rho$  and false otherwise. Given a literal  $l$ ,  $\rho$  may only go through  $\text{Test}(l)$  if  $\nu(l)$  is true; because  $\rho$  covers  $q_f$ , this proves that  $\nu \models \phi$ . ◀

► **Remark 9.** In [11], the authors prove NP-completeness of COVER in a similar model, but with a leader: in the NP-hardness reduction, the leader make non-deterministic decisions about the values of the variable. This argument does not hold in the leaderless case.

### 3.2 Interesting Restrictions

Although all the problems defined above are NP-complete, they are sometimes tractable under appropriate restrictions on the protocols. We will consider two restrictions on the protocols. The first one is having  $\text{dim} = 1$ , *i.e.*, a single register. The second restriction is the *uninitialized case* where processes are not allowed to read the initial value  $d_0$  from the registers. Formally, a protocol  $\mathcal{P}$  is *uninitialized* if its set of transitions  $\Delta$  does not contain an action reading symbol  $d_0$ : in uninitialized protocols, it is structurally impossible to read from an unwritten register. One might object that forbidding transitions that read  $d_0$  contradicts the intuition that, when a process reads from a register, it does not know whether the value is initial or not; one could settle the issue by considering that reading  $d_0$  sends processes to a sink state. The uninitialized setting tends to yield better complexity than the general, initialized case, see for example [1, Section 7].

Of course, for PRP, the formula itself always makes the problem NP-hard.

► **Proposition 10.** *PRP for roundless register protocols is NP-hard even with  $\text{dim} = 1$  and the register uninitialized.*

### 3.3 Tractability of COVER and DNF-PRP under Restrictions

In this subsection, we prove that COVER is solvable in PTIME when the protocol is uninitialized or when  $\text{dim}$  is fixed and that DNF-PRP is solvable in PTIME when  $\text{dim} = 1$ .

In [11, Theorem 9.2], uninitialized COVER is proved to be PTIME-complete; their approach, based on languages, is quite different from the one presented here. Our approach, similar to the one presented in [9, Algorithm 1] in the setting of reconfigurable broadcast networks, is to compute the set of coverable states using a simple *saturation* technique, a fixed-point computation over the set of states.

When registers are initialized, the saturation technique breaks down as it may be that some states are coverable but not in the same execution, as they require registers to lose their initial value in different orders (see the notion of first-write order developed in [6] for more development on this in a round-based setting). However, in the initialized case with a fixed number of registers, one can iterate over every such order and COVER is tractable as well.

► **Proposition 11.** *COVER for roundless register protocols is PTIME-complete either when the registers are uninitialized or when  $\text{dim}$  is fixed.*

For DNF-PRP, we provide a PTIME algorithm in the more restrictive case of  $\text{dim} = 1$ .

► **Proposition 12.** *DNF-PRP for roundless register protocols with  $\text{dim} = 1$  is in PTIME.*

**Proof sketch.** We give here the proof for TARGET. Our algorithm shares similarities with [12, page 41] for broadcast protocols, although it is more complex because of the persistence of symbols in the register.

First, we have a polynomial reduction from initialized TARGET with  $\text{dim} = 1$  to uninitialized TARGET with  $\text{dim} = 1$ . It proceeds as follows. Consider the graph  $G = (Q, E)$  when  $(q_1, q_2) \in E$  when there exists  $(q_1, \text{read}(\mathbf{d}_0), q_2) \in \Delta$ . Let  $I \subseteq Q$  the set of states that are reachable in  $G$  from  $Q_0$ . The reduction simply replaces  $Q_0$  by  $I$  as set of initial states.

Any (abstract) execution  $\rho : \sigma_0 \xrightarrow{*} \langle q_f, \mathbf{d}_f \rangle$ , called *synchronizing execution*, can be rearranged into  $\rho_+ : \sigma_0 \xrightarrow{*} \langle S, \mathbf{d} \rangle$  and  $\rho_- : \langle S, \mathbf{d} \rangle \xrightarrow{*} \langle q_f, \mathbf{d}_f \rangle$  where  $S$  contains all states that appear in  $\rho$ . Additionally, we can make  $\rho_-$  start with a write action (there is a transition in  $\rho$  that writes  $\mathbf{d}$ ). To obtain the decomposition,  $\rho_+$  mimics  $\rho$  but does not empty any state, and  $\rho_-$  mimics  $\rho$  but from a configuration with more states. We compute the maximum such set  $S$  by iteratively deleting states that cannot appear in any synchronizing execution. Let

$$\mathcal{C}(\mathcal{P}) := \max\{S \subseteq Q \mid \exists \mathbf{d} \in \mathbf{D}, \exists \sigma_0 \in \text{Init}_a, \sigma_0 \xrightarrow{*} \langle S, \mathbf{d} \rangle\}$$

$$\mathcal{BC}(\mathcal{P}) := \max\{S \subseteq Q \mid \forall \mathbf{d} \in \mathbf{D}, \exists \mathbf{d}' \in \mathbf{D}, \langle S, \mathbf{d}' \rangle \xrightarrow{*} \langle q_f, \mathbf{d}_f \rangle\}$$

Both maxima exist as the sets are non-empty ( $Q_0$  is included in the first set and  $q_f$  is in the second set) and they are stable by union (concatenate the corresponding executions). Intuitively,  $\mathcal{C}(\mathcal{P})$  corresponds to the set of coverable sets, and  $\mathcal{BC}(\mathcal{P})$  to the set of backward coverable states. In the decomposition  $\rho_+ : \sigma_0 \xrightarrow{*} \langle S, \mathbf{d} \rangle$ ,  $\rho_- : \langle S, \mathbf{d} \rangle \xrightarrow{*} \langle q_f, \mathbf{d}_f \rangle$ ,  $\rho_+$  is a witness that  $S \subseteq \mathcal{C}(\mathcal{P})$  and  $\rho_-$  that  $S \subseteq \mathcal{BC}(\mathcal{P})$  (because  $\rho_-$  starts with a write action, for every  $\mathbf{d}' \in \mathbf{D}$  one has  $\langle S, \mathbf{d}' \rangle \xrightarrow{*} \langle q_f, \mathbf{d}_f \rangle$ ).

$\mathcal{C}(\mathcal{P})$  and  $\mathcal{BC}(\mathcal{P})$  can be computed in polynomial time. For  $\mathcal{C}(\mathcal{P})$ , we use a saturation technique. For  $\mathcal{BC}(\mathcal{P})$ , we work backwards: a symbol is read before it is written. We start with  $S := \{q_f\}$ . Until a fixpoint for  $S$  is reached, we do the following. We iterate on  $\mathbf{D}$ , trying to pick the symbol that was in the register before  $S$  could be reached. For each  $\mathbf{d} \in \mathbf{D}$ , we saturate  $S$  with backward transitions reading  $\mathbf{d}$ , then check if  $\mathbf{d}$  can be written by a transition ending in  $S$ . If not, we backtrack by removing states that were just added.

The algorithm iteratively removes from  $\mathcal{P}$  states that are not in  $\mathcal{C}(\mathcal{P}) \cap \mathcal{BC}(\mathcal{P})$ . Indeed, states that are not in  $\mathcal{C}(\mathcal{P}) \cap \mathcal{BC}(\mathcal{P})$  cannot appear in any synchronizing execution. If it ends up with  $Q(\mathcal{P}) = \emptyset$ , then there is no synchronizing execution and the algorithm rejects. If it ends up with  $\mathcal{C}(\mathcal{P}) = \mathcal{BC}(\mathcal{P}) = Q(\mathcal{P}) \neq \emptyset$ , then applying the definitions of  $\mathcal{C}(\mathcal{P})$  and  $\mathcal{BC}(\mathcal{P})$  gives a synchronizing execution, and the algorithm accepts. ◀

It is unknown whether the previous result still holds when  $\text{dim}$  is fixed to a value greater than 1. The case  $\text{dim} = 1$  is particularly easy because writing to the register completely erases its content.

Unlike COVER, TARGET and therefore DNF-PRP are not tractable under the uninitialized hypothesis. For TARGET, one cannot add fresh processes at no cost, since the fresh processes would eventually have to get to  $q_f$ . For example, if a register  $r$  can only be written from a given state  $q$ , the last process to leave  $q$  will fix the value in register  $r$ .

► **Proposition 13.** *TARGET for uninitialized roundless register protocols is NP-hard.*

	COVER	TARGET	DNF-PRP	PRP
General case	NP-complete (Prop. 8)	NP-complete (Prop. 8)	NP-complete (Prop. 8)	NP-complete (Prop. 8)
Uninitialized	PTIME-complete (Prop. 11)	NP-complete (Prop. 8 & 13)	NP-complete (Prop. 8 & 13)	NP-complete (Prop. 8 & 10)
$\text{dim} = 1$ (one register)	PTIME-complete (Prop. 11)	PTIME-complete (Prop. 12 & 11)	PTIME-complete (Prop. 12 & 11)	NP-complete (Prop. 8 & 10)

■ **Figure 3** Summary of complexity results for roundless register protocols.

## 4 Round-based Register Protocols

We now extend the previous model to a round-based setting. The model and semantics are the same as in [6], however we consider a more general problem than COVER. Thus, the abstract semantics developed here differs from [6].

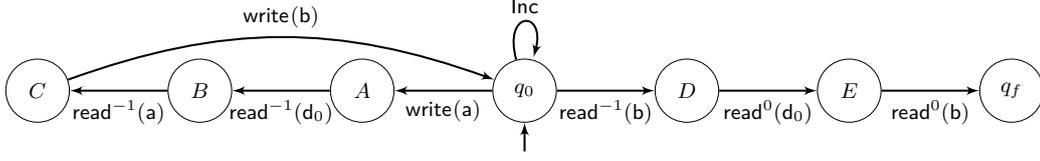
### 4.1 Definitions

In round-based settings, there is a fresh set of  $\text{dim}$  registers at each round, and each process has its own private round value that starts at 0 and never decreases. Processes may only read from and write to registers of nearby rounds.

► **Definition 14** (Round-based register protocols). *A round-based register protocol is a tuple  $\mathcal{P} = \langle Q, Q_0, \text{dim}, D, d_0, v, \Delta \rangle$  where*

- $Q$  is a finite set of states with a distinguished subset of initial states  $Q_0 \subseteq Q$ ;
- $\text{dim} \in \mathbb{N}$  is the number of shared registers per round;
- $D$  is a finite data alphabet with an initial symbol  $d_0$ ;
- $v$  is the visibility range;
- $\Delta \subseteq Q \times \mathcal{A} \times Q$  is the set of transitions, where  $\mathcal{A} = \{\text{read}_\alpha^{-i}(d) \mid i \in [0, v], \alpha \in [1, \text{dim}], d \in D\} \cup \{\text{write}_\alpha(d) \mid \alpha \in [1, \text{dim}], d \in D \setminus \{d_0\}\} \cup \{\text{Inc}\}$  is the set of actions.

Read actions specify the round of the register:  $\text{read}_\alpha^{-i}(d)$  means, for a process at round  $k$ , “read  $d$  from register  $\alpha$  of round  $k-i$ ”. A process at round  $k$  may only write to the registers of round  $k$ . The Inc action increments the round of a process.



■ **Figure 4** An example of round-based register protocol.

Let  $\text{rg}_k[\alpha]$  denote the register  $\alpha$  of round  $k$ . The set of registers of round  $k$  is written  $\text{Reg}_k$ , and we let  $\text{Reg} = \bigcup_{k \in \mathbb{N}} \text{Reg}_k$ . The size of a protocol is  $|\mathcal{P}| = |Q| + |D| + |\Delta| + \nu + \dim$ . A given process is described by its state and round, formalized by a pair  $(q, k) \in Q \times \mathbb{N}$  called *location*. Let  $\text{Loc} := Q \times \mathbb{N}$  denote the set of locations. A *concrete configuration* describes the number of processes in each location along with the value of each register. Formally, a *concrete configuration* is a pair  $\langle \mu, \vec{d} \rangle$  with  $\mu \in \mathbb{N}^{\text{Loc}}$  such that  $0 < \sum_{(q,k) \in \text{Loc}} \mu(q,k) < \infty$  and  $\vec{d} \in D^{\text{Reg}}$ . For  $\gamma = \langle \mu, \vec{d} \rangle$ , we write  $\text{loc}(\gamma) := \mu$  and  $\text{data}(\gamma) := \vec{d}$ . Again, we write  $\Gamma$  for the set of concrete configurations. The set of initial configurations is  $\text{Init}_c := \{\gamma \in \Gamma \mid \text{data}(\gamma) = \vec{d}_0^{\text{Reg}} \text{ and } \forall (q,k) \notin Q_0 \times \{0\}, \text{loc}(\gamma)(q,k) = 0\}$ .

A *move* is a pair  $\theta \in \Delta \times \mathbb{N}$ :  $\text{move}(\delta, k)$  expresses that transition  $\delta$  is taken by a process at round  $k$ ; we write  $\text{Moves} := \Delta \times \mathbb{N}$  for the set of all moves. A move  $\theta$  has *effect* on round  $k$  when  $\theta$  is at round  $k$  or  $\theta$  is an increment at round  $k-1$ . We define a step as follows: for  $\theta = ((q, a, q'), k) \in \text{Moves}$ ,  $\gamma \xrightarrow{\theta} \gamma'$  when  $(q, k) \in \text{loc}(\gamma)$  and:

- if  $a = \text{read}_{\alpha}^{-i}(d)$ ,  $\text{loc}(\gamma') = (\text{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k)\}$ ,  $\text{data}(\gamma)(\text{rg}_{k-i}[\alpha]) = d$  and  $\text{data}(\gamma') = \text{data}(\gamma)$ ;
- if  $a = \text{write}_{\alpha}(d)$ ,  $\text{loc}(\gamma') = (\text{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k)\}$ ,  $\text{data}(\gamma')(\text{rg}_k[\alpha]) = d$  and for all  $r \neq \text{rg}_k[\alpha]$ ,  $\text{data}(\gamma')(r) = \text{data}(\gamma)(r)$ ;
- if  $a = \text{Inc}$ ,  $\text{loc}(\gamma') = (\text{loc}(\gamma) \ominus \{(q, k)\}) \oplus \{(q', k+1)\}$  and  $\text{data}(\gamma') = \text{data}(\gamma)$ .

A step is *at round  $k$*  when the corresponding move is of the form  $(\delta, k)$ . Note that action  $\text{read}_{\alpha}^{-i}(d)$  is only possible for processes at rounds  $k \geq i$ . The notions of execution, of reachability and of coverability are defined as in the roundless case.

► **Example 15.** Consider the round-based protocol  $\mathcal{P}$  from Figure 4, with  $\dim = 1$ ,  $\nu = 1$ ,  $Q_0 = \{q_0\}$  and  $D = \{d_0, a, b\}$ . In this protocol, state  $q_f$  cannot be covered. By contradiction, consider an execution  $\pi : \gamma_0 \xrightarrow{*} \gamma$  with  $\gamma_0 \in \text{Init}_c$  and  $\text{loc}(\gamma)(q_f, k) > 0$  for some  $k \in \mathbb{N}$ . We have that, at some point in  $\pi$ ,  $(E, k)$  is populated and  $b$  is in  $\text{rg}[k]$ . Therefore, some process went from  $(A, k)$  to  $(B, k)$ , which implies that  $\text{rg}[k]$  lost value  $d_0$  before  $\text{rg}[k-1]$ ; this in turn implies that  $\pi$  does not send any process to  $(E, k)$  which is a contradiction.  $\lrcorner$

Since round-based register protocols enjoy the same monotonicity properties as roundless register protocols, we define the same non-counting abstraction. Note that this abstraction differs from the one in [6] which was designed specifically for COVER. The set of *abstract configurations* is  $\Sigma := 2^{\text{Loc}} \times D^{\text{Reg}}$ ; the abstract semantics are defined as in Subsection 2.3. Again,  $\sigma \xrightarrow{\delta} \sigma'$  if and only if there exist  $\gamma, \gamma' \in \Gamma$ ,  $\gamma \xrightarrow{\delta} \gamma'$  and  $\text{abst}(\gamma) = \sigma$ ,  $\text{abst}(\gamma') = \sigma'$ . All the properties of Subsection 2.3 apply to round-based abstract semantics. In particular, we have the soundness and completeness of the abstraction:

► **Proposition 16** (Soundness and completeness of the abstraction). *For all  $L \subseteq \text{Loc}$ ,  $\vec{d} \in D^{\text{Reg}}$ :*

$$(\exists \gamma \in \text{Reach}_c(\text{Init}_c) : \text{supp}(\gamma) = L, \text{data}(\gamma) = \vec{d}) \iff (\exists \sigma \in \text{Reach}_a(\text{Init}_a) : \text{loc}(\sigma) = L, \text{data}(\sigma) = \vec{d}).$$

## 4.2 Presence Reachability Problem

COVER is extended to round-based protocols by asking whether some reachable configuration has a process on  $q_f$  on some round  $k$ , and TARGET by asking whether some reachable configuration has no process on states  $q \neq q_f$  on any round  $k$ . Formally, one asks whether there exists  $\gamma \in \text{Reach}_c(\text{Init}_c)$  such that  $\gamma \models \psi$  where  $\psi = “\exists k \in \mathbb{N}, (q, k) \in \text{loc}(\gamma)”$  for COVER and  $\psi = “\forall k \in \mathbb{N}, \forall q \neq q_f, (q, k) \notin \text{loc}(\gamma)”$  for TARGET. We will now extend roundless PRP to round-based PRP, where the formula is allowed to have non-nested quantification over rounds.

Presence constraints are first-order formulas (quantifying over the rounds) without any nested quantifiers. Formally:

- a *term* is of the form  $m$  or  $k+m$  with  $m \in \mathbb{N}$  and  $k$  a free variable;
- an *atomic proposition* is either of the form “ $(q, t)$  populated” with  $t$  a term and  $q \in Q$  or of the form “ $\text{rg}_t[\alpha]$  contains  $d$ ” with  $t$  a term,  $\alpha \in [1, \text{dim}]$  and  $d \in D$ ;
- a *literal* is either an atomic proposition or the negation of an atomic proposition;
- a *proposition* is a Boolean combination of atomic propositions that has at most one free variable;
- an *atomic presence constraint* is either a *closed* proposition (no free variables), or of the form “ $\exists k \phi$ ” or “ $\forall k \phi$ ” where  $\phi$  is a proposition with  $k$  as a free variable.

Finally, a *presence constraint* is a Boolean combination of atomic presence constraints.

► **Example 17.** “ $(\exists k (q_2, k)$  populated)  $\vee$  ( $\forall k ((q_0, k+2)$  populated)  $\wedge$   $\text{rg}_1[1]$  contains a)” is an example of presence constraint.

Let  $\gamma := ((q_0, 0) \oplus (q_1, 1), d_0^{\text{Reg}})$  with  $q_0 \neq q_1$ ,  $\text{dim} = 1$ . One has  $\gamma \models (\text{rg}_0[1]$  contains  $d_0) \wedge (\exists k (q_1, k+1)$  populated) but  $\gamma \not\models \forall k (((q_0, k)$  populated)  $\vee$   $\neg((q_1, k)$  populated)). ◻

We define the *round-based presence reachability problem* (round-based PRP):

ROUND-BASED PRP

**Input:** A round-based register protocol  $\mathcal{P}$ , a presence constraint  $\psi$

**Question:** Does there exist  $\gamma \in \text{Reach}_c(\text{Init}_c)$  such that  $\gamma \models \psi$ ?

► **Example 18.** Consider  $\mathcal{P}$  from Example 15. If  $\psi := \exists k, (q_f, k)$  populated, then  $(\mathcal{P}, \psi)$  is a negative instance of round-based PRP. If  $\psi' := \exists k, ((E, k)$  populated)  $\wedge$   $((E, k+1)$  populated), then  $(\mathcal{P}, \psi')$  is also negative. However, if  $\psi'' := ((E, 2)$  populated)  $\wedge$   $[\forall k, (\text{rg}[k+1]$  contains  $b) \vee (\text{rg}[k+1]$  contains  $d_0)]$ , then  $(\mathcal{P}, \psi'')$  is positive: a witness execution sends a process to  $(B, 1)$ , writes  $a$  to  $\text{rg}[0]$  then  $b$  to  $\text{rg}[1]$  and finally sends a process from  $(q_0, 2)$  to  $(E, 2)$ . ◻

COVER and TARGET for round-based register protocols are special cases of PRP. The following lower bound hence applies to all these problems:

► **Proposition 19** ([6, Theorem 23]). *COVER for round-based register protocols is PSPACE-hard, even in the uninitialized case with  $v = 0$  and  $\text{dim} = 1$ .*

Note that, in the round-based setting,  $\text{dim} = 1$  means one register *per round*, therefore still an unbounded number of registers.  $v = 0$  means that a process can only interact with registers of its current round. The previous proposition implies that all problems considered in Figure 3 are PSPACE-hard when working with round-based protocols. In [6], COVER for round-based register protocols is shown to be PSPACE-complete. In the rest of this paper, we establish that the more general round-based PRP lies in the same complexity class:

► **Theorem 20.** *Round-based PRP is PSPACE-complete.*

## 5 A Polynomial-Space Algorithm for Round-Based PRP

In this section, we provide a polynomial-space algorithm for round-based PRP. Thanks to Savitch's theorem, it suffices to find a non-deterministic polynomial-space algorithm. To do so, one wants to guess an execution that reaches a configuration satisfying the presence constraint. However, as shown in [6, Proposition 13], one may need, at a given point along such an execution, the number of active rounds to be exponential (an active round being informally a round on which something has already happened and something else is yet to happen). Thus, storing the execution step by step in polynomial space seems hard; instead, our algorithm will guess the execution round by round. To do this, we define the notion of footprint, which represents the projection of an execution onto a narrow window of rounds.

Thanks to Proposition 7, round-based PRP can be studied directly in the abstraction. In the rest of the paper, all configurations and executions are implicitly abstract.

### 5.1 Footprints

Let  $j \leq k$ . We write  $\text{Loc}[j, k]$  for the set of locations at rounds  $\max(j, 0)$  to  $k$ ; similarly, we write  $\text{Reg}[j, k]$  for the set of registers of rounds  $\max(j, 0)$  to  $k$ . A *local configuration* on (rounds)  $[j, k]$  is an element of  $2^{\text{Loc}[j, k]} \times \text{D}^{\text{Reg}[j, k]}$ . The set of local configurations on  $[j, k]$  is written  $\Sigma[j, k]$ . Given  $\sigma \in \Sigma$ , the local configuration  $\text{local}[j, k](\sigma)$  is obtained by removing from  $\sigma$  all information that is not about rounds  $j$  to  $k$ . Note that local configurations are local with respect to the rounds, and not with respect to processes.

Given  $\lambda, \lambda' \in \Sigma[j, k]$  and a move  $\theta$ , we write  $\lambda \xrightarrow{\theta} \lambda'$  when there exist two configurations  $\sigma$  and  $\sigma'$  such that  $\sigma \xrightarrow{\theta} \sigma'$ ,  $\text{local}[j, k](\sigma) = \lambda$  and  $\text{local}[j, k](\sigma') = \lambda'$ . In practice:

- if  $\theta$  is a move with no effect on rounds  $j$  to  $k$ , then  $\lambda \xrightarrow{\theta} \lambda'$  if  $\lambda = \lambda'$ ;
- if  $\theta = ((q, \text{Inc}, q'), j-1)$  then  $\lambda \xrightarrow{\theta} \lambda'$  holds with no condition that  $(q, j-1)$  is populated in  $\lambda$ , since  $j-1$  is outside of  $[j, k]$ ;
- if  $\theta = ((q, \text{read}_\alpha^{-b}(\text{d})), l)$  with  $l-b < j$  (read from register of round  $< j$ ), there is no condition on the content of the register.

A *footprint* on (rounds)  $[j, k]$  corresponds to the projection of an execution on rounds  $[j, k]$ . Formally, it is an alternating sequence  $\lambda_0, \theta_0, \lambda_1, \dots, \theta_{m-1}, \lambda_m$  where for all  $i \in [0, m]$ ,  $\lambda_i \in \Sigma[j, k]$  and for all  $i \leq m-1$ ,  $\lambda_i \xrightarrow{\theta_i} \lambda_{i+1}$  and  $\lambda_i \neq \lambda_{i+1}$ .

Let  $\rho = \sigma_0, \theta_0, \sigma_1, \dots, \theta_{m-1}, \sigma_m$  be an execution. The *footprint of  $\rho$  on (rounds)  $[j, k]$* , written  $\text{footprint}[j, k](\rho)$ , is the footprint on  $[j, k]$  obtained from  $\rho$  by replacing  $\sigma_i$  by  $\lambda_i = \text{local}[j, k](\sigma_i)$  and then removing all useless steps  $\lambda_i \xrightarrow{\theta} \lambda_{i+1}$  with  $\lambda_i = \lambda_{i+1}$  (by merging  $\lambda_i$  and  $\lambda_{i+1}$ , so  $\text{footprint}[j, k](\rho)$  can be shorter than  $\rho$ ). Similarly, for  $[j', k'] \supseteq [j, k]$  and  $\tau$  a footprint on  $[j', k']$ , define the *projection footprint  $[j, k](\tau)$*  by the footprint obtained by replacing each local configuration in  $\tau$  by its projection on  $[j, k]$  and removing useless steps.

The following result provides a sufficient condition for a sequence of footprints to be seen as projections of a single common execution.

► **Lemma 21.** *Let  $K \in \mathbb{N}$ ,  $(\tau_k)_{k \leq K}$  and  $(T_k)_{k \leq K-1}$  such that:*

- *for all  $k \leq K$ ,  $\tau_k$  is a footprint on  $[k-v+1, k]$ ,*
- *for all  $k \leq K-1$ ,  $T_k$  is a footprint on  $[k-v+1, k+1]$ ,*
- *for all  $k \leq K-1$ ,  $\text{footprint}[k-v+1, k](T_k) = \tau_k$ ,*
- *for all  $k \leq K-1$ ,  $\text{footprint}[k-v+2, k+1](T_k) = \tau_{k+1}$ .*

*There exists an execution  $\rho$  such that, for all  $k \leq K$ ,  $\text{footprint}[k-v+1, k](\rho) = \tau_k$ .*

## 5.2 A Polynomial-Space Algorithm for Round-Based PRP

The algorithm guesses the witness execution footprint by footprint, and stops when the presence constraint is satisfied. Algorithm 1 provides the skeleton of this procedure. For the sake of simplicity, we suppose that  $v \geq 1$ . If  $v = 0$ , we artificially increase  $v$  to 1.

■ **Algorithm 1** Non-deterministic algorithm for round-based PRP.

---

```

1 Input: A PRP instance  $(\mathcal{P}, \psi)$ 
2  $E, U, C \leftarrow \emptyset$  ;
3  $\tau \leftarrow \epsilon$  ; // dummy footprint on rounds  $[-v, -1]$ 
4 Guess the initial set  $I \subseteq Q_0$  of populated states at round 0 ;
5  $\text{NDInit}(E, U, C)$  ;
6 for  $k$  from 0 to  $+\infty$  do
7   Guess  $T$  a footprint on  $[k-v, k]$  such that  $\text{footprint}[k-v, k-1](T) = \tau$  ;
8   Check that  $T$  is consistent with the initial configuration ;
9    $\lambda \leftarrow$  last configuration in  $T$  ;
10   $\text{NDComputeIteration}(E, U, C, \lambda)$  ;
11  if  $\text{TestPresenceConstraint}(E, U, C, \lambda)$  then Accept ;
12   $\tau \leftarrow \text{footprint}[k-v+1, k](T)$  ;
```

---

For all  $k \in \mathbb{N}$ , let  $\tau_k$  be the value of  $\tau$  at the end of iteration  $k$  and  $T_k$  the value of  $T$  guessed at iteration  $k+1$ . Thanks to Lemma 21, if the algorithm reaches the end of iteration  $K$  then there exists an execution  $\rho$  whose projection on  $[k-v, k-1]$  is  $\tau_k$  for every  $k \leq K$ .

Handling the round-based presence constraint is technical, so we hide it in functions  $\text{NDInit}$ ,  $\text{NDComputeIteration}$  and  $\text{TestPresenceConstraint}$ , whose pseudocode can be found in Algorithm 2. We guess why  $\psi$  is true by guessing satisfied *atomic propositions* of three types: existentially quantified on the round (*i.e.*, of the form “ $\exists k \phi$ ” where  $\phi$  has no quantifiers and only  $k$  as free variable) which we put in  $E$ ; universally quantified on the round (*i.e.*, of the form “ $\forall k \phi$ ” where  $\phi$  has no quantifiers and only  $k$  as free variable) which we put in  $U$ ; with no quantifier (*i.e.*, of the form “ $\phi$ ” where  $\phi$  has no quantifiers and no free variables) which we put in  $C$ . Formulas in  $C$  refer to constant rounds and are checked at these rounds only. Formulas in  $U$  are checked at every round. For formulas in  $E$ , the algorithm guesses at which round the formula is true. See Appendix B.3 of the full version for more detailed explanations. Our algorithm is correct with respect to round-based PRP:

► **Proposition 22.**  $(\mathcal{P}, \psi)$  is a positive instance of round-based PRP if and only if there exists an accepting computation of Algorithm 1 on  $(\mathcal{P}, \psi)$ .

The integer constants in the presence constraint  $\psi$  are encoded in unary, like the visibility range  $v$ . These two hypotheses are reasonable since practical examples typically use constants of small value (*e.g.*, 1). Under these hypotheses, we obtain a polynomial spatial bound on the size of footprints of a well-chosen witness execution, which in turn gives a polynomial spatial bound for the algorithm:

► **Proposition 23.** Algorithm 1 works in space  $O(|\psi|^3 + |Q|^2 (v+1)^2 \log(\dim |D|))$ .

Finally, we need to discuss the termination of the algorithm. According to the pigeonhole principle, after an exponential number of iterations, the elements stored in memory repeat from a previous iteration and we can stop the computation. One can thus use a counter, encoded in polynomial space, to count iterations and return a decision when the counter

reaches its largest value. Thanks to the space bounds from Proposition 23, correctness from Proposition 22 and the stopping criterion, our algorithm decides round-based PRP in non-deterministic polynomial space, proving Theorem 20.

■ **Algorithm 2** The functions at **Line 5**, **Line 10** and **Line 11** of Algorithm 1.

---

```

1 Function NDInit( $E, U, C$ ) :
   | /* Sets containing what needs to be checked:  $U$  and  $E$  contain
   |   respectively universally and existentially quantified atomic
   |   presence constraints,  $C$  contains closed literals          */
2   Guess  $X \subseteq \text{PosOrNeg}(\text{APC}(\psi))$  s.t.  $\psi$  is true when all APCs in  $X$  are true ;
3   for  $P$  in  $X$  do
4     for  $\phi$  closed atomic proposition in  $P$  do
5       | /*  $\phi$  refers to constant rounds only          */
6       | if  $\phi$  guessed to be true then Add  $\phi$  to  $C$  ; Replace  $\phi$  by true in  $P$  ;
7       | else Add  $\neg\phi$  to  $C$  ; Replace  $\phi$  by false in  $P$  ;
8       | if  $P$  is a closed proposition then
9         | Check that  $P$  is true with guessed values of atomic propositions ;
10        | if  $P$  universal then Add  $P$  to  $U$  ;
11        | if  $P$  existential then Add  $P$  to  $E$  ;
12 Function NDComputeIteration( $E, U, C, \lambda$ ) :
13   for " $\forall l \phi$ " in  $U$  do
14     | Guess  $\mathcal{L} \subseteq \text{PosOrNeg}(\text{AP}(\phi[l \leftarrow k]))$  s.t.  $\phi[l \leftarrow k]$  is true when all literals in  $\mathcal{L}$ 
15     | are true ;
16     | Add all literals in  $\mathcal{L}$  to  $C$  ;
17   for " $\exists l \phi$ " in  $E$  do
18     | if  $\phi[l \leftarrow k]$  guessed to be true then
19     |   Guess  $\mathcal{L} \subseteq \text{PosOrNeg}(\text{AP}(\phi[l \rightarrow k]))$  s.t.  $\phi[l \leftarrow k]$  is true when all literals in
20     |    $\mathcal{L}$  are true ;
21     |   Add all literals in  $\mathcal{L}$  to  $C$  ; Remove " $\exists l \phi$ " from  $E$  ;
22   for  $\phi$  in  $C$  about round  $k$  do
23     | //  $\phi$  is of the form (negation of) " $(q, k)$  populated", or (negation
24     |   of) " $\text{rg}_k[\alpha]$  contains  $d$ "
25     | Check that  $\phi$  is satisfied in  $\lambda$  ; Remove  $\phi$  from  $C$  ;
26 Function TestPresenceConstraint( $E, U, C, \lambda$ ) :
27   if  $E \neq \emptyset$  then return false ;
28   for  $\phi \in C$  or " $\forall l \phi$ " in  $U$  do
29     | if  $\langle \emptyset, d_0^{\text{Reg}} \rangle \not\models \phi$  then
30     |   return false ; // Execution cannot stop at round  $k$ 
31   return true ;

```

---

## References

- 1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Rojin Rezvan. Parameterized verification under TSO is PSPACE-complete. *Proc. ACM Program. Lang.*, 4(POPL):26:1–26:29, 2020. doi:10.1145/3371094.
- 2 James Aspnes. Fast deterministic consensus in a noisy environment. *Journal of Algorithms*, 45(1):16–39, 2002. doi:10.1016/S0196-6774(02)00220-1.

- 3 A. R. Balasubramanian, Lucie Guillou, and Chana Weil-Kennedy. Erratum to parameterized analysis of reconfigurable broadcast networks, 2022. URL: <https://www.model.in.tum.de/~weilkenn/erratum-fossacs22.pdf>.
- 4 A. R. Balasubramanian, Lucie Guillou, and Chana Weil-Kennedy. Parameterized analysis of reconfigurable broadcast networks (long version). *CoRR*, abs/2201.10432, 2022. arXiv: 2201.10432.
- 5 A. R. Balasubramanian and Chana Weil-Kennedy. Reconfigurable broadcast networks and asynchronous shared-memory systems are equivalent. In *GandALF 2021*, volume 346, pages 18–34, 2021. doi:10.4204/EPTCS.346.2.
- 6 Nathalie Bertrand, Nicolas Markey, Ocan Sankur, and Nicolas Waldburger. Parameterized safety verification of round-based shared-memory systems. In *ICALP 2022*, pages 113:1–113:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.113.
- 7 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.
- 8 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 106:1–106:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.106.
- 9 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *FSTTCS 2012*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.289.
- 10 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *STACS 2014*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.
- 11 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *Journal of the ACM*, 63(1):10:1–10:48, 2016. doi:10.1145/2842603.
- 12 Paulin Fournier. *Parameterized verification of networks of many identical processes*. PhD thesis, University of Rennes 1, France, 2015. URL: <https://tel.archives-ouvertes.fr/tel-01355847>.
- 13 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.
- 14 Matthew Hague. Parameterised pushdown systems with non-atomic writes. In *FSTTCS 2011*, volume 13 of *LIPIcs*, pages 457–468. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.457.
- 15 Michel Raynal and Julien Stainer. A simple asynchronous shared memory consensus algorithm based on omega and closing sets. In *CISIS 2012*, pages 357–364. IEEE Computer Society, 2012. doi:10.1109/CISIS.2012.198.

